# A Dip in the Reservoir: Maintaining Sample Synopses of Evolving Datasets

Rainer Gemulla Wolfgang Lehner
Technische Universität Dresden
01099 Dresden, Germany
{gemulla,lehner}@inf.tu-dresden.de

Peter J. Haas
IBM Almaden Research Center
San Jose, California, USA
phaas@us.ibm.com

## ABSTRACT

Perhaps the most flexible synopsis of a database is a random sample of the data; such samples are widely used to speed up processing of analytic queries and data-mining tasks, enhance query optimization, and facilitate information integration. In this paper, we study methods for incrementally maintaining a uniform random sample of the items in a dataset in the presence of an arbitrary sequence of insertions and deletions. For "stable" datasets whose size remains roughly constant over time, we provide a novel sampling scheme, called "random pairing" (RP) which maintains a bounded-size uniform sample by using newly inserted data items to compensate for previous deletions. The RP algorithm is the first extension of the almost 40-year-old reservoir sampling algorithm to handle deletions. Experiments show that, when dataset-size fluctuations over time are not too extreme, RP is the algorithm of choice with respect to speed and sample-size stability. For "growing" datasets, we consider algorithms for periodically "resizing" a bounded-size random sample upwards. We prove that any such algorithm cannot avoid accessing the base data, and provide a novel resizing algorithm that minimizes the time needed to increase the sample size.

## 1. INTRODUCTION

Because of its flexibility, sampling is widely used for quick approximate query answering [1, 5, 10, 9, 12, 24], statistics estimation [11, 26], data stream processing [13, 17, 27], data mining [3, 16, 18, 20], and data integration [2, 14, 15, 22]. Uniform random sampling, in which all samples of the same size are equally likely, is the most fundamental of the available database sampling schemes. Uniform sampling is ubiquitous in applications: most statistical estimators—as well as the confidence-bound formulas for these estimators—assume an underlying uniform sample. Thus uniformity is a must if it is not known in advance how the sample will be used. Uniform sampling is also a building block for more complex sampling schemes, such as stratified sampling. Methods for producing uniform samples are therefore key to modern database systems.

In the simplest setting, the basic task is to compute a uniform sample from a dataset that is stored on disk, such as a table in a relational database management system (RDBMS) or a repository of XML documents. In general, there are two alternative approaches to computing such a sample. First, the sample may be materialized on the fly as it is needed. In the setting of commercial RDBMS, Haas and König [12] have shown that there is a trade-off between the uniformity of a sampling scheme and the cost of computing it. Even if some degree of non-uniformity is acceptable, online sample materialization can still be too expensive. Moreover, it is often quite acceptable to use the same sample several times, in order to answer a set of queries or perform multiple analysis tasks. Taking advantage of this fact, an alternative approach [4, 8, 10, 11, 17] amortizes the cost of sampling over multiple uses by initially materializing a sample from a dataset and then incrementally maintaining the "sample synopsis" over time.

Incremental sample maintenance is a very powerful technique, because the abstract notion of the underlying "dataset" can be interpreted very broadly in applications. Indeed, the dataset can actually be an arbitrary view, e.g., over the result of an arbitrary SQL query. Samples over views are particularly good candidates for incremental maintenance, because producing such samples on the fly can require very expensive base-data accesses. For example, most relational operators are not interchangeable with sampling [5, 24], so that in most cases the sampling operator cannot be moved to the leaves of the query tree. Olken and Rotem [24, 25] pioneered methods for incremental maintenance of sample views in relational databases; these methods synthesize traditional view-maintenance techniques with database sampling algorithms. The idea is to, in effect, compute the "delta" (set of insertions, updates, and deletions) to the view as the underlying tables are updated and then apply general sample-maintenance methods to the resulting sequence of view modifications. Although computation of the deltas requires access to the base data, this expense is smoothly spread out over time, providing for fast query response. Also observe that the full view need never be materialized if only the sample is of interest, thereby saving space as well as time. The main deficiency of existing techniques for maintaining sample views is that they require expensive base-data accesses over and above those needed to compute the deltas.

This paper provides new methods for incrementally maintaining a uniform random sample of an evolving dataset. We assume that the sample-maintenance component intercepts data insertion and deletion requests[1] on their way to the dataset, and maintains the sample locally. In this setting, the main challenges in sample maintenance are (1) to enforce statistical uniformity in the presence of arbitrary insertions and deletions, (2) to avoid accesses to the base data to the extent possible, because such accesses are typically expensive, and (3) to keep the sample size as stable as possible, avoiding oversized or undersized samples.

We distinguish between "stable" datasets whose size (but not necessarily composition) remains roughly constant over time and "growing" datasets in which insertions occur more frequently than deletions over the long run. The former setting is typical of transactional database systems and databases of moving objects; the latter setting is typical of data warehouses in which historical data accumulates. For stable datasets, it is highly desirable from a systems point of view to ensure that the sample size stays below a specified upper bound, so that memory for the sample can be allocated initially, with no unexpected memory overruns occurring later on. Moreover, once memory has been allocated for the sample, the sample size should be kept as close to the upper bound as possible in order to maximize the statistical precision of applications that use the sample. That is, we want to use the allotted space efficiently. For growing data sets, maintaining a bounded sample is of limited practical interest. Over time, such a sample represents an increasingly small fraction of the dataset. Although a diminishing sampling fraction may not be a problem for tasks such as estimating a population sum, many other tasks—such as estimating the number of distinct values of a specified population attribute—require the sampling fraction to be bounded from below. The goal for a growing data set is therefore to grow the sample in a stable and efficient manner, guaranteeing an upper bound on the sample size at all times and using the allotted space efficiently.

The best known method for incrementally maintaining a sample in the presence of a stream of insertions to the dataset is the classical "reservoir sampling" algorithm [21, 23], which maintains a simple random sample of a specified size. One deficiency of this method is that it cannot handle deletions, and the most obvious modifications for handling deletions either yield procedures for which the sample size systematically shrinks to 0 over time or which require expensive base-data accesses.[2] The other main deficiency is that the class of pure insertion streams—for which reservoir sampling is designed—results in growing datasets as discussed above; thus the usefulness of the bounded reservoir sample tends to diminish over time. Surprisingly, although reservoir sampling has been around for almost 40 years, the algorithm apparently has never been extended to deal with either deletions or growing datasets. In this paper we provide the first such extensions of reservoir sampling.

In more detail, we address the challenges of incremental sample maintenance as follows.

1. For stable datasets, we provide a new sampling scheme, called "random pairing" (RP), that maintains a bounded uniform sample in the presence of arbitrary insertions and deletions, without requiring expensive base-data accesses. RP can be viewed as a generalization of both classical reservoir sampling and the "passive" stream-sampling algorithm of Babcock, et al. [1]. RP is as least as fast as any other known uniform sampling scheme, and strictly faster than any scheme that accesses the base data. The sample sizes produced by RP are more stable than those produced by any other algorithm that does not access base data and, provided that fluctuations in the dataset size are not too extreme, are as stable as those produced by expensive algorithms that require base-data accesses. Thus, if the dataset size is reasonably stable over time, RP is the algorithm of choice for incrementally maintaining a bounded uniform sample.

2. For growing datasets, we initiate the study of algorithms for periodically "resizing" a bounded-size random sample upwards, thereby allowing an ever-increasing sample size while at all times avoiding uncontrollably large samples. We prove that any such algorithm cannot avoid accessing the base data, and provide a novel resizing algorithm that permits the sample to grow over time while always enforcing a user-controlled upper bound on the sample size. We show how to set the algorithm parameters to optimally balance the time required to access the base data and the time needed to subsequently enlarge the sample using newly inserted data.

The remainder of this paper is organized as follows. In Section 2, we review existing algorithms that are pertinent to incremental sample maintenance, and relate them to our new techniques. Section 3 contains a description and correctness proof of the RP algorithm. Section 4 describes our resizing algorithm, and develops approximate cost formulas for this algorithm that can be used to tune the key algorithm parameter. In Section 5, we report results from an empirical performance study of the new and existing sample-maintenance algorithms; we also assess the accuracy of our approximate cost model for the resizing algorithm. We conclude in Section 6.

## 2. UNIFORM SAMPLING SCHEMES

In this section, we describe the sampling problem more precisely and give an overview of various new and existing sampling schemes. Following [4], call a sampling scheme *uniform* if the probability $\mathscr{P}(S;R)$ that the scheme produces sample $S$ when applied to dataset $R$ satisfies $\mathscr{P}(S;R) = \mathscr{P}(S';R)$ whenever $|S| = |S'|$. That is, all samples of the same size are equally likely to be produced. We say that $S$ "is a uniform sample from $R$" if $S$ is produced from $R$ using a uniform sampling scheme. We restrict attention throughout to sampling without replacement.

We consider a (possibly infinite) set $\mathcal{T} = \{ t_1, t_2, \dots \}$ of unique, distinguishable *items* that are inserted into and deleted from the dataset $R$ over time; for example, $\mathcal{T}$ might correspond to a finite set of IP addresses, an infinite sequence of text documents, or an infinite sequence of sales transactions over an evolving selection of products. In general, items that are deleted may be subsequently re-inserted. Without loss of generality, we assume throughout that $R$ is initially empty. Thus we consider an infinite sequence of

[1]We do not consider updates explicitly, since an update to the dataset can be trivially handled by updating the value of the corresponding sample element, if present.

[2]A common approach is to periodically recompute the sample from scratch [11].

transactions $\gamma = (\gamma_1, \gamma_2, \ldots)$, where each transaction $\gamma_i$ is either of the form $+t_k$, which corresponds to the insertion of item $t_k$ into $R$, or of the form $-t_k$, which corresponds to the deletion of item $t_k$ from $R$. We restrict attention to "feasible" sequences such that (i) at any time point, an item appears at most once in the dataset (so that the dataset is a true set and not a bag) and (ii) $\gamma_n = -t_k$ only if item $t_k$ is in the dataset just prior to the processing of the $n$th transaction. Our goal is to ensure that, after each transaction is processed, $S$ is a uniform sample from $R$. We assume throughout that, as is usual in practice, the sequence $\gamma$ of insertions and deletions to the data is oblivious to the behavior of the sampling algorithm.

We first discuss two classical uniform schemes, Bernoulli sampling and reservoir sampling, that underlie all of the other sampling methods. We then discuss schemes that are appropriate for stable datasets and growing datasets. Finally, we discuss the relationship of these schemes to some recent work on "distinct-value" (DV) sampling.

## 2.1 Two Classical Schemes

**Bernoulli Sampling**: In the Bernoulli sampling scheme with sampling rate $q$, denoted $\mathrm{BERN}(q)$, each inserted item is included in the sample with probability $q$ and excluded with probability $1-q$, independent of the other items. For a dataset $R$, the sample size follows the binomial distribution $\mathrm{BINOM}(|R|, q)$, so that $P\{\,|S| = k\,\} = \binom{|R|}{k} q^k (1-q)^{|R|-k}$ for $k = 0, 1, \ldots, |R|$. Although the sample size is random, samples having the same size are equally likely, so that the scheme is indeed uniform, as defined previously. The main advantages of Bernoulli sampling are simplicity and ease of parallelization; the main disadvantage is the uncontrollable variability of the sample size. Indeed, the sample can be as large as $|R|$, so there is no effective upper bound.

**Reservoir Sampling (RS)**: This uniform scheme maintains a random sample of fixed size $M$, given a sequence of insertions. The procedure, as described in McLeod et al. [23], is as follows. Include the first $M$ items into the sample. For each successive insertion into the dataset, include the inserted item into the sample with probability $M/|R|$, where $|R|$ is the size of the dataset just after the insertion; an included item replaces a randomly selected item in the sample. Vitter [28] significantly reduced the computational costs of RS by devising a method to directly generate the (random) number of arriving items to skip between consecutive sample inclusions, thereby avoiding the need to "flip a coin" for each item. Efficient reservoir schemes that handle very large disk-based samples are provided in [8, 17].

## 2.2 Schemes for Stable Datasets

**Stream-sampling methods**: Babcock et al. [1] have proposed several sampling schemes for obtaining a fixed-size uniform random sample from a moving window over a data stream. This setting corresponds to the special case in which each deletion from the dataset is immediately followed by an insertion, and these algorithms do not directly generalize to arbitrary sequences of insertions and deletions. The most pertinent of the algorithms in [1] is the "passive" algorithm. This algorithm first obtains a uniform sample from the initial window. Whenever an item in the sample is deleted from the window, the corresponding newly inserted item takes the place of the deleted item in the sample. Brown and Haas [4] provide "approximate" stream-sampling algorithms for a warehouse in which "data partitions" are rolled in and out. The idea is to create samples of the data partitions that "shadow" the full partitions as they move through the warehouse. Again, these algorithms do not generalize to arbitrary, item-wise insertions and deletions, but the "merging" algorithms in [4] can potentially be used to parallelize the new algorithms in the current paper.

**Correlated acceptance-rejection (CAR)**: By adapting the CAR algorithm of Olken and Rotem [25] to our setting, we obtain a method for maintaining uniform random samples in the presence of arbitrary insertions and deletions; this method requires access to the base data, however. Our version of CAR actually maintains a uniform sample with replacement, i.e., each item in the dataset may appear more than once in the sample. Whenever an item is inserted into the dataset, CAR generates a random number $N$ from the binomial distribution $\mathrm{BINOM}(M, 1/|R|)$ and replaces $N$ random items of the current sample by $N$ copies of the new item. Whenever an item is deleted from the dataset, CAR replaces each occurrence of this item by a random item drawn from the population. We obtain the final uniform sample without replacement by removing duplicates; thus the gross sample size must be larger than $M$ to compensate for duplicate removal, and there is no effective lower bound on the sample size.

**Correlated acceptance-rejection without replacement (CARWOR)**: This simple variant of the CAR algorithm executes standard RS at each insertion. Whenever an item is deleted from the sample $S$, CARWOR replaces it by a random item from $R \setminus S$. Although CARWOR maintains the sample size at its largest possible value, the algorithm relies on frequent, expensive accesses to base data.

**Reservoir sampling with recomputation (RSR)**: As mentioned previously, RS is designed to deal only with insertions. The simplest modification is to execute RS as usual at each insertion. At each deletion from the dataset, we check whether the item is in the sample; if so, we remove it and continue RS with a smaller sample size. The obvious problem with this approach is that the sample size decreases monotonically to zero. We therefore modify this approach using a device as in Gibbons et al. [11]: as soon as the sample size falls below a prespecified lower bound, recompute it from scratch using, for example, sequential sampling [29]. Clearly, this approach (also called the "backing sample" method) does not yield a stable sample size, and it requires repeated access to the base data. In spite of these deficiencies, RSR has been the sampling scheme of choice for bounded-size uniform sampling.

**Bernoulli sampling with purging (BSP)**: This technique combines Bernoulli sampling with an idea proposed by Gibbons et al. [10], and can handle insertions and deletions without accessing base data.[3] The idea is to use $\mathrm{BERN}(q)$ sampling—allowing deletions, as in the $\mathrm{MBERN}(q)$ scheme described in Section 2.3 below—and to purge the sample every time it exceeds the upper bound. Starting with $q = 1$, we decrease $q$ at every purge step. With $q'$ being the new value of $q$, the sample is subsampled using $\mathrm{BERN}(q'/q)$ sampling. This procedure is repeated until the sample size has fallen below $M$. The choice of $q'$ is challenging: on the one hand, if $q'$ is chosen small with respect to $q$, the sample size drops significantly below the upper bound in expecta-

[3]We note that the specific sampling algorithms given in [10] do not produce uniform samples; see [4].

tion. On the other hand, a high value of $q'$ leads to frequent purges, thereby reducing performance. Note that BSP does not maintain a true BERN($q$) sample, because the sample size is bounded. Due to the difficulty of choosing $q$ and, as discussed in the sequel, instability in the sample sizes, this algorithm can be difficult to use in practice.

**Random pairing (RP)**: Our new RP algorithm, described in Section 3, maintains a bounded-size uniform sample in the presence of arbitrary insertions and deletions without requiring access to the base data. As shown in our experiments, the RP algorithm produces samples that are significantly larger (i.e., more space efficient) and more stable than those produced by BSP, at lower cost.

## 2.3 Schemes for Growing Datasets

**Modified Bernoulli Sampling**: This uniform sampling scheme, denoted MBERN($q$), is the simplest scheme for dealing with a growing data set. The MBERN($q$) scheme treats each insertion identically to the ordinary BERN($q$) scheme. Whenever an item is removed from the dataset, it is also removed from the sample, if present. As with BERN($q$), the sample size is binomially distributed, and is $100q\%$ of the dataset size on average. Note that we do not consider the use of MBERN($q$) for stable datasets because it cannot guarantee an upper bound on the sample size (nor can it guarantee a lower bound).

**Resizing**: Our novel resizing method can be used with any stable-dataset sampling scheme. In this way we can grow the sample as the dataset grows, while guaranteeing an upper bound on the sample size at each time point. Resizing can even be used in conjunction with MBERN($q$) sampling to increase the sampling rate $q$; see Section 4.2.

## 2.4 Distinct-Value Sampling

To our knowledge, the only other sampling methods that handle arbitrary sequences of insertions and deletions are the two DV-sampling algorithms recently proposed in [6, 7]. These two algorithms were designed for datasets in which items can appear multiple times; i.e., datasets that are bags, not sets. The algorithms sample uniformly from the set of distinct items of a dataset, and also provide the number of occurrences for each sampled value (or a high-accuracy approximation thereof). In our setting, where each item in the dataset occurs only once, random sampling and DV-sampling coincide, so we could attempt to adapt the DV-sampling algorithms to our purpose.

Both DV-sampling algorithms make use of a data structure which—with some success probability $p$—maintains a single item chosen randomly from the set of distinct items. To maintain multi-item samples, multiple instances of the data structure are stored, so that sampling is with replacement. The two schemes [6, 7] coincide when adapted to our setting: each data structure consists of the *sum* of the items (treated as integers) inserted into it, and a *counter* of the number of inserted items. The basic idea is that only a random fraction of the items added into or deleted from the dataset affect a data structure. In more detail, the adapted DV-sampling algorithm (call it DVS) would make use of a set of independent random hash functions, one for each instance of the data structure. The range of each hash function is $\{0, \dots, D-1\}$, where $D$ is the average size of the dataset. An item $t$ affects a data structure only if the corresponding hash function satisfies $h(t) = 0$. If a data structure contains exactly one item, then it "succeeds," and returns this item as a random sample of size 1; otherwise, it fails.

The probability $p$ of a success is approximately equal to $e^{-1}$ when $D$ is large, so that DVS would exploit only 1/3 of the available memory, at best. Not only is the space efficiency low, but the computational costs are also extremely high. For example, with 1 million copies of the data structure, DVS requires 10 trillion hash operations to insert 10 million tuples into the sample, which takes hours even with the fastest available hash functions. Schemes such as RP, which are tailored for unique-item sampling, can perform such an insertion in a matter of minutes. For these reasons, we focus on the uniform sampling schemes outlined in the previous sections.

## 3. RANDOM PAIRING

To motivate the idea behind the random-pairing scheme, we first consider an "obvious" passive algorithm for maintaining a bounded uniform sample $S$ of a dataset $R$. The algorithm avoids accessing base data by making use of new insertions to "compensate" for previous deletions. Whenever an item is deleted from the data set, it is also deleted from the sample, if present. Whenever the sample size lies at its upper bound $M$, the algorithm handles insertions identically to RS; whenever the sample size lies below the upper bound and an item is inserted into the dataset, the item is also inserted into the sample. Although simple, this algorithm is unfortunately incorrect, because it fails to guarantee uniformity. To see this, suppose that, at some stage, $|S| = M < |R| = N$. Also suppose that an item $t^-$ is then deleted from the dataset $R$, directly followed by an insertion of $t^+$. Denote by $S'$ the sample after these two operations. If the sample is to be truly uniform, then the probability that $t^+ \in S'$ should equal $M/N$, conditional on $|S| = M$. Since $t^- \in S$ with probability $M/N$, it follows that

$$\begin{aligned} &P\{\, t^+ \in S' \,\} \\ &= P\{\, t^- \in S,\ t^+ \text{ included} \,\} + P\{\, t^- \notin S,\ t^+ \text{ included} \,\} \\ &= \frac{M}{N} \cdot 1 + \Big(1 - \frac{M}{N}\Big) \cdot \frac{M}{N} > \frac{M}{N}, \qquad (1) \end{aligned}$$

conditional on $|S| = M$. Thus an item inserted just after a deletion has an overly high probability of being included in the sample. The basic idea behind RP is to avoid the foregoing problem by including an inserted item into the sample with a probability less than 1 when the sample size lies below the upper bound. The key question is how to select the inclusion probability to ensure uniformity.

## 3.1 Algorithm Description

In the RP scheme, every deletion from the dataset is eventually compensated by a subsequent insertion. At any given time, there are 0 or more "uncompensated" deletions; the number of uncompensated deletions is simply the difference between the cumulative number of insertions and the cumulative number of deletions. The RP algorithm maintains a counter $c_1$ that records the number of uncompensated deletions in which the deleted item was in the sample (so that the deletion also decremented the sample size by 1). The RP algorithm also maintains a counter $c_2$ that records the number of uncompensated deletions in which the deleted item was not in the sample (so that the deletion did not affect

**Algorithm 1** Random pairing

```
 1: $c_1$: no. of deletions which have been in the sample
 2: $c_2$: no. of deletions which have not been in the sample
 3: $k$: skip counter for reservoir sampling (initialized to 1)
 4: $M$: upper bound on sample size
 5: $R$, $S$: dataset and sample, respectively
 6: SKIP(): reservoir-sampling skip function as in [28]
 7:    [SKIP$(m, r) = 0$ if $r < m$]
 8:
 9: INSERT($t$):
10: if $c_1 + c_2 = 0$ then    // execute plain reservoir sampling
11:    $k \leftarrow k - 1$
12:    if $k = 0$ then
13:       if $|S| < M$ then
14:          insert $t$ into $S$
15:       else
16:          overwrite a randomly selected element of $S$ with $t$
17:       end if
18:       $k \leftarrow$ SKIP$(M, |R|) + 1$
19:    end if
20: else                          // execute random-pairing step
21:    if RANDOM() $< \frac{c_1}{c_1+c_2}$ then
22:       $c_1 \leftarrow c_1 - 1$
23:       insert $t$ into $S$
24:    else
25:       $c_2 \leftarrow c_2 - 1$
26:    end if
27: end if
28:
29: DELETE($t$):
30: if $t \in S$ then
31:    $c_1 \leftarrow c_1 + 1$
32:    remove $t$ from $S$
33: else
34:    $c_2 \leftarrow c_2 + 1$
35: end if
```

the sample). Clearly, $d = c_1 + c_2$ is the total number of uncompensated deletions.

The algorithm works as follows. Deletion of an item is handled by removing the item from the sample, if present, and by incrementing the value of $c_1$ or $c_2$, as appropriate. If $d = 0$, i.e., there are no uncompensated deletions, then insertions are processed as in standard RS, using Vitter's optimizations [28]. If $d > 0$, then we flip a coin at each insertion step, and include the incoming insertion into the sample with probability $c_1/(c_1 + c_2)$; otherwise, we exclude the item from the sample. We then decrease either $c_1$ or $c_2$, depending on whether the insertion has been included into the sample or not. The complete algorithm is given as Algorithm 1.

Conceptually, whenever an item is inserted and $d > 0$, the item is paired with a randomly selected uncompensated deletion, called the "partner" deletion. The inserted item is included into the sample if its partner was in the sample at the time of its deletion, and excluded otherwise. The probability that the partner was in the sample is $c_1/(c_1 + c_2)$. For the purpose of the algorithm, it is not necessary to keep track of the identity of the random partner; it suffices to maintain the counters $c_1$ and $c_2$. Note that if we repeat the calculation in (1) using RP, we now have $P\left\{ t^- \notin S,\ t^+ \text{ included} \right\} = 0$, and we obtain the desired result $P\left\{ t^+ \in S' \right\} = M/N$.

## 3.2 An Example

The RP algorithm with $M = 2$ is illustrated in Figure 1. The figure shows all possible states of the sample, along with the probabilities of the various state transitions. The example starts after $i = 2$ items have been inserted into an empty dataset, i.e., the sample coincides with $R$. The insertion of item $t_3$ leads to the execution of a standard RS step since there are no uncompensated deletions. This step has three possible outcomes, each equally likely. Next, we remove items $t_2$ and $t_3$ from both the dataset and the sample. Thus, at $i = 5$, there are two uncompensated deletions. The insertion of $t_4$ triggers the execution of a *pairing step*. Item $t_4$ is conceptually paired with either $t_3$ or $t_2$—these scenarios are denoted by *a*) and *b*) respectively—and each of these pairings is equally likely. Thus $t_4$ compensates its partner, and is included in the sample if and only if the partner was in the sample prior to its deletion. This pairing step amounts to including $t_4$ with probability $c_1/(c_1 + c_2)$ and excluding $t_4$ with probability $c_2/(c_1 + c_2)$, where the values of $c_1$ and $c_2$ depend on which path is taken through the tree of possibilities. A pairing step is also executed at the insertion of $t_5$, but this time there is only one uncompensated deletion left: $t_2$ in scenario *a*) or $t_3$ in scenario *b*). The probability of seeing a given sample at a given time point is computed by multiplying the probabilities along the path from the "root" at the far left to the node that represents the sample. Observe that the sampling scheme is indeed uniform: at each time point, all samples of the same size are equally likely to have been materialized.

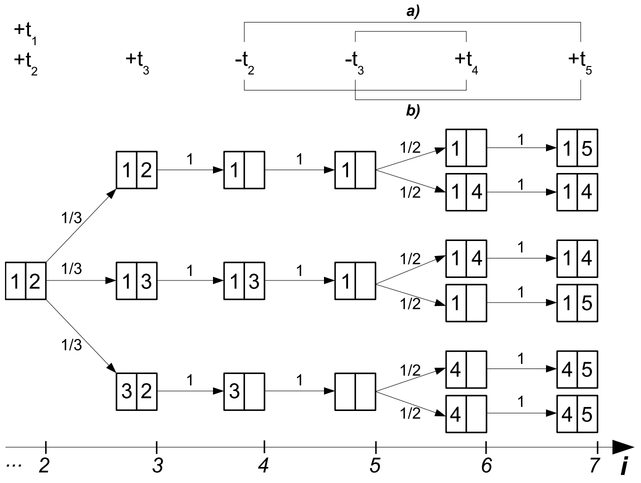


**Figure 1: Random pairing (possible outcomes & probability)**

## 3.3 Correctness and Sample-Size Properties

In this section we formally establish the uniformity property of the RP scheme with upper bound $M$ ($\geq 1$) and also derive formulas for the mean and variance of the sample size. To establish uniformity, we actually prove a slightly stronger result that implies uniformity. Denote by $R_n$ the dataset and by $S_n$ sample after the $n$th processing step, i.e., after processing transaction $\gamma_n$. Also denote by $c_{1,n}$ and $c_{2,n}$ the value of the counters $c_1$ and $c_2$ after the $n$th step, and set $d_n = c_{1,n} + c_{2,n}$. Finally, set $u_n = \min(M, |R_n|)$, $v_n = \min(M, \max_{1 \leq j \leq n}|R_j|) = \min(M, |R_n| + d_n)$, and $l_n = \max(0, u_n - d_n)$; in light of (4) below, it can be seen that $u_n$ and $l_n$ are the largest and smallest possible sample sizes after the $n$th step, and $v_n$ is the largest possible sample size so far (thus $u_n \leq v_n$). Without loss of generality, we restrict attention to sequences that start with an insertion into an empty dataset.

THEOREM 1. *For any feasible sequence $\gamma$ of insertions and deletions, there exist numbers $\{ p_n(k) \colon n \geq 1$ and $k \geq 0 \}$, depending on $\gamma$, such that*

$$P\{ S_n = A \} = p_n(|A|) \tag{2}$$

*for $A \subseteq R_n$ and $n \geq 1$. Moreover,*

$$\frac{p_n(k)}{p_n(k-1)} = \frac{v_n - k + 1}{d_n - v_n + k}. \tag{3}$$

*for $n \geq 1$ and $k \in \{ l_n + 1, l_n + 2, \ldots, u_n \}$.*

It follows from (2) that, at each step, any two samples of the same size are equally likely to be produced, so that the RP algorithm is indeed a uniform sampling scheme.

PROOF. Clearly, we can take $p_n(k) = 0$ for $n \geq 1$ and $k \notin \{ l_n, l_n + 1, \ldots, u_n \}$. Fix a sequence of insertions and deletions, and observe that the sample size decreases whenever $c_1$ increases, and increases whenever $c_1$ decreases (subject to the constraint $|S| \leq M$.) It follows directly that

$$c_{1,n} = v_n - |S_n| \tag{4}$$

for $n \geq 1$. The proof now proceeds by induction on $n$. The assertions of the theorem clearly hold for $n = 1$, so suppose for induction that the assertions hold for values $1, 2, \ldots, n-1$. There are two cases to consider. First, suppose that step $n$ corresponds to the insertion of an item $t$, and consider a subset $A \subseteq R_n$ with $|A| = k$, where $l_n \leq k \leq u_n$. If $d_{n-1} = 0$, then $d_n = 0$ and $l_n = u_n$, so that (3) holds vacuously, and the correctness proof for standard reservoir sampling—see, e.g., [13]—establishes the assertion in (2). So assume in the following that $d_{n-1} > 0$. If $t \in A$, then, using (4), we have

$$\begin{aligned} P\{ S_n = A \} &= P\{ S_{n-1} = A - \{t\},\ t \text{ included} \} \\ &= p_{n-1}(k-1)\frac{c_{1,n-1}}{d_{n-1}} = p_{n-1}(k-1)\frac{v_{n-1} - k + 1}{d_{n-1}}. \end{aligned}$$

If $t \notin A$, then

$$\begin{aligned} P\{ S_n = A \} &= P\{ S_{n-1} = A,\ t \text{ ignored} \} \\ &= p_{n-1}(k)\frac{d_{n-1} - v_{n-1} + k}{d_{n-1}}, \end{aligned} \tag{5}$$

so that, if $A \neq \emptyset$, then

$$P\{ S_n = A \} = p_{n-1}(k-1)\frac{v_{n-1} - k + 1}{d_{n-1}}. \tag{6}$$

Here (6) follows from (5) and an inductive application of (3). This establishes the first assertion of the theorem with

$$p_n(k) = \begin{cases} p_{n-1}(k-1)\frac{v_{n-1}-k+1}{d_{n-1}} & \text{if } \max(l_n, 1) \leq k \leq u_n; \\ p_{n-1}(0)\frac{d_{n-1}-v_{n-1}}{d_{n-1}} & \text{if } k = l_n = 0; \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

To establish the second assertion of the theorem, apply (7) and then inductively apply (3), making use of the fact that—since $d_{n-1} > 0$ and an item is inserted at step $n$—we have $d_n = d_{n-1} - 1$ and $v_n = v_{n-1}$. Now suppose that step $n$ corresponds to the deletion of an item $t$, and again consider a subset $A \subseteq R_n$ with $|A| = k \in \{ l_n, l_n + 1, \ldots, u_n \}$. Observe that

$$\begin{aligned} P\{ S_n = A \} &= P\{ S_{n-1} = A \} + P\{ S_{n-1} = A \cup \{t\} \} \\ &= p_{n-1}(k) + p_{n-1}(k+1), \end{aligned}$$

which establishes the first assertion of the theorem with $p_n(k) = p_{n-1}(k) + p_{n-1}(k+1)$ for $l_n \leq k \leq u_n$. Since $d_n = d_{n-1} + 1$ and $v_n = v_{n-1}$, we then have

$$\begin{aligned} \frac{p_n(k)}{p_n(k-1)} &= \frac{p_{n-1}(k) + p_{n-1}(k+1)}{p_{n-1}(k-1) + p_{n-1}(k)} \\ &= \frac{\big(p_{n-1}(k)/p_{n-1}(k-1)\big) + \big(p_{n-1}(k+1)/p_{n-1}(k-1)\big)}{1 + \big(p_{n-1}(k)/p_{n-1}(k-1)\big)} \\ &= \frac{v_{n-1} - k + 1}{d_{n-1} - v_{n-1} + k + 1} = \frac{v_n - k + 1}{d_n - v_n + k} \end{aligned}$$

for $l_n < k \leq u_n$, where we have again inductively used (3). Thus the second assertion of the theorem holds and the proof is complete. □

Observe that the RP scheme reduces to the "passive" algorithm of [1] if applied to a fixed-width moving window over a data stream. If there are no deletions, the RP scheme reduces to standard RS.

The following result gives some basic statistical properties of the sample size at any given time point.

THEOREM 2. *Let $d$ be the number of uncompensated deletions. Then the expected value and variance of the sample size $|S|$ are given by $E[|S|] = v|R|/(|R| + d)$ and*

$$\mathrm{Var}[|S|] = \frac{dv(|R| + d - v)|R|}{(|R| + d)^2(|R| + d - 1)}$$

*where $v = \min(M, |R| + d)$.*

For example, suppose we sample $100,000$ items from a dataset consisting of $10,000,000$ items (1%). If we delete $100,000$ items, the sample size is $99,000$ in expectation and has a standard deviation of 31.31 items, thus $98{,}968 \leq |S| \leq 99{,}032$ with high probability.

Observe that if $d = 0$, so that there are no uncompensated deletions, or if $|R| + d \leq M$, so that the sample coincides with the population, then $E[|S|] = \min(M, |R|)$ and $\mathrm{Var}[|S|] = 0$, i.e., the sample size is deterministic.

PROOF. First suppose that $d = 0$. Because compensated deletions do not affect the sample size, it follows that the sample size is the same as if there never had been any deletions, that is, $\min(M, |R|)$. Now suppose that $d > 0$. Without loss of generality, we can assume that the last $d$ operations have all been deletions from the dataset. Clearly, the sample size prior to the first of the $d$ deletions is $v = \min(M, |R| + d)$. Let $X$ be the number of the $d$ deleted items that were sample items. Then $X$ has a hypergeometric distribution, i.e., $P\{ X = k \} = \binom{v}{k}\binom{|R|+d-v}{d-k}/\binom{|R|+d}{d}$ for $k = 0, 1, \ldots, \min(d, v)$, where we use the convention that $\binom{0}{0} = 1$ and $\binom{0}{k} = 0$ for $k > 0$. Note that $P\{ X = d \} = 1$ when $|R| + d \leq M$. The mean and variance of the sample size correspond to $E[v - X] = v - E[X]$ and $\mathrm{Var}[v - X] = \mathrm{Var}[X]$, respectively, and the desired result now follows directly from well known properties of the hypergeometric distribution [19, p. 238]. □

## 4. RESIZING SAMPLES

The discussion so far has focused on stable datasets, and therefore on sampling algorithms that guarantee a fixed upper bound on the sample size. We now shift attention to growing datasets. As mentioned previously, modified Bernoulli sampling can be used to maintain a sample whose size grows with the dataset, but such a sampling scheme cannot control the maximum sample size. We therefore consider the problem of maintaining a sample with an upper bound that is periodically increased.

### 4.1 A Negative Result

The RP algorithm can maintain a bounded sample without needing to access the base data. One might hope that there exist algorithms for resizing a sample that similarly do not need to access the base data. Theorem 3 below shows that such algorithms cannot exist.

In general, we consider algorithms that start with a uniform sample $S$ of size at most $M$ from a dataset $R$ and—after some finite (possibly zero) number of arbitrary insertion/deletion transactions on $R$—produce a uniform sample $S'$ of size $M'$ from the resulting modified dataset $R'$, where $M < M' < |R|$. In general, we also allow the algorithm to access the base dataset $R$. For example, a trivial resizing scheme ignores the transactions altogether, immediately discards $S$, and creates a fresh sample $S'$ by resampling $R$.

Theorem 3. *There exists no resizing algorithm that can avoid accessing the base dataset $R$.*

Proof. Suppose to the contrary that such an algorithm exists, and consider the case in which the transactions on $R$ consist entirely of insertions. Fix a set $A \subseteq R'$ such that $|A| = M'$ and $A$ contains $M+1$ elements of $R$; such a set can always be constructed under our assumptions. Because the hypothesized algorithm produces uniform samples of size $M'$ from $R'$, we must have $P\{\, S' = A \,\} > 0$. But clearly $P\{\, S' = A \,\} = 0$, since $|S| \le M$ and, by assumption, no further elements of $R$ have been added to the sample. Thus we have a contradiction, and the result follows. $\square$

### 4.2 A Resizing Algorithm

We now provide a method for resizing a bounded-size sample; this method is given as Algorithm 2.

Suppose that the initial sample size is $|S| = M$ and the target sample size is $M' > M$, where $M, M' < |R|$. The basic idea is as follows. In phase 1, the algorithm converts the sample to a BERN($q$) sample, possibly accessing base data in the process; we discuss the choice of $q$ in the following section. In phase 2, the algorithm uses Bernoulli sampling (with deletions allowed) to increase the sample size to the target value $M'$. At this point, bounded-size sampling resumes, using the new upper bound $M'$. In more detail, the algorithm generates a random variable $U$ having a BINOM($|R|, q$) distribution, which represents the initial Bernoulli sample size. The algorithm uses as many items from $S$ as possible to make up the Bernoulli sample, accessing base data only if $U > |S|$. If the initial sample size $U$ exceeds the target size $M'$, then the algorithm simply materializes a sample of size $M'$ from $R$ and terminates, in effect taking an immediate subsample of size $M'$ from a Bernoulli sample of size $U$. In phase 2, the algorithm increases the sample to the desired size by using MBERN($q$) sampling.

**Algorithm 2** Sample Resizing

```
1: $M$: initial sample size
2: $M'$: target sample size ($M' > M$)
3: $q$: Bernoulli sampling parameter
4: $R$: initial dataset
5: $S$: initial sample with $|S| = M$
6:
7: PHASE 1:
8: generate $U$ from BINOM($|R|, q$) distribution
9: **if** $U \le M$ **then**
10:     $S \leftarrow$ uniform subsample of size $U$ from $S$
11:     go to Phase 2
12: **else if** $M < U < M'$ **then**
13:     $V \leftarrow$ uniform sample of size $U - M$ from $R \setminus S$
14:     $S \leftarrow S \cup V$
15:     go to Phase 2
16: **else if** $U \ge M'$ **then**
17:     $V \leftarrow$ uniform sample of size $M' - M$ from $R \setminus S$
18:     $S \leftarrow S \cup V$
19:     return $S$
20: **end if**
21:
22: PHASE 2:
23: **while** $|S| < M'$ **do**
24:     wait for request                          // *insert or delete*
25:     **if** request = "insert item" **then**
26:         insert item into $S$ with probability $q$
27:     **else**
28:         remove item from $S$ if present
29:     **end if**
30: **end while**
31: return $S$
```

More formally, denote by $S^*$ the effective sample at the end of phase 1. Also denote by $S_k$ and $R_k$ ($k \ge 0$) the elements in the sample and the dataset after $k$ insertion/deletion transactions have occurred in phase 2. Note that $S_0 = S^*$ if $U < M'$ and $S_0$ is a size-$M'$ uniform subsample from $S^*$ if $U \ge M'$. Finally, denote by $L$ ($\ge 0$) the random number of transactions that occur during phase 2. The following theorem asserts the correctness of the resizing algorithm.

Theorem 4. *Given that $k \le L$, where $k \ge 0$, the set $S_k$ is a uniform sample from $R_k$.*

Proof. (Sketch) As mentioned previously, the distribution of $U$ in phase 1 is identical to the distribution of the sample size of a BERN($q$) sample of $R$. The subsampling step ($U \le M$) and the union step (otherwise) both maintain the uniformity of $S$, so that $S^*$ is a BERN($q$) sample from $R$. Using this fact, it can then be shown that every probability of the form $P\{\, S_k = A \mid k \le L \,\}$ with $A \subseteq R_k$ depends on $A$ only through $|A|$, and the desired result follows. For example, when $k \ge 1$ and all phase 2 transactions are insertions, fix a set $A \subseteq R_k$ comprising exactly $i$ elements of $R_0$ and $j$ elements of $R_k \setminus R_0$, where $i + j < M'$ and $j \le k$. Then

$$P\{\, S_k = A, k \le L \,\} = P\{\, S_k = A \,\}$$
$$= q^i (1-q)^{|R|-i} q^j (1-q)^{k-j} = q^{|A|}(1-q)^{|R|+k-|A|},$$

and $P\{\, S_k = A \mid k \le L \,\} = P\{\, S_k = A, k \le L \,\} / P\{\, k \le L \,\}$ depends on $A$ only through $|A|$. $\square$

We assume that the dataset is "locked" during phase 1, so that the process of incoming insertion and deletion requests is temporarily suspended. The value of the parameter $q$ therefore determines both 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.

As a final observation, if we are using an MBERN($q$) sampling scheme to deal with a growing dataset, then we can execute phase 1 with parameter $q' > q$ to transition from MBERN($q$) sampling to MBERN($q'$) sampling. In the following sections, however, we focus on the use of the resizing algorithm with algorithms that produce bounded-size samples.

## 4.3 Performance Analysis: Choosing q

To provide guidance in choosing the value of $q$, we have developed exact and approximate cost models of the resizing process. Due to lack of space, we describe only the approximate cost model and resulting choice of $q$: the approximate procedure is much easier to implement and, as shown by our experiments (Section 5), highly accurate. We also focus primarily on the case in which there are no deletions; the general case is discussed briefly at the end of the section.

We separately analyze the costs (in units of elapsed time) for phase 1 and phase 2. During phase 1, the algorithm obtains $N(U)$ items from $R \setminus S$, where $N(u) = (\min(u, M') - M)^+$ for $u \geq 0$, with $x^+ = \max(x, 0)$. These $N(U)$ 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. Because of the acceptance-rejection step, the (random) number $B_i$ of base-data accesses required to obtain the $i$th item has a geometric distribution with failure probability $(M+i-1)/|R|$, from which it follows [19, p. 201] that $E[B_i] = |R|/(|R| - M - i + 1)$. As our first approximation, we assume that $U = u_q$ with probability 1, where $u_q$ is the closest integer to $|R|q = E[U]$; our motivation is that the coefficient of variation $(\mathrm{Var}[U]/E^2[U])^{1/2}$ is of order $O(|R|^{-1/2})$, and $|R|$ is typically very large. Thus the total number of base accesses during phase 1 is approximately $B = B_1 + B_2 + \cdots + B_{N(u_q)}$, and $E[B] = |R|\, H\big(|R| - M - N(u_q) + 1, |R| - M\big)$, where $H(n, m) = \sum_{i=n}^{m} 1/i$. Using the standard approximation $H(n, m) \approx \ln(m/n)$ for large $m, n$ and supposing that each base access takes $t_a$ time units, we find that the expected time to execute phase 1 is approximately given by $\hat{T}_1(q) = t_a|R| \ln\big[(|R| - M)/\big(|R| - M - N(|R|q)\big)\big]$.

In phase 2, the resizing algorithm executes a random number $L$ of Bernoulli trials until the sample size reaches the target value $M'$. Specifically, given that $U = u$, we have $L \equiv 0$ if $u \geq M'$, since phase 2 is not actually executed in this case; if $u < M'$, then $L - (M' - u)$, the number of Bernoulli rejections before the sample size reaches $M'$, has a negative binomial distribution. Again assuming that $P\{\, U = u_q \,\} = 1$ and appealing to [19, p. 199], we have $E[L] \approx (M' - u_q)^+/q$. Suppose that the average time from the completion of a Bernoulli trial to the completion of the next Bernoulli trial is $t_b$; this quantity primarily reflects the time between successive insertion/deletion requests and is assumed to be essentially constant. Then the expected time to execute phase 2 is approximately $\hat{T}_2(q) = t_b(M' - |R|q)^+/q$, and the expected total time required to resize a sample is approximately equal to $\hat{T}(q) = \hat{T}_1(q) + \hat{T}_2(q)$.

We now choose $q = \hat{q}^*$, where $\hat{q}^*$ minimizes the function $\hat{T}$. To compute $\hat{q}^*$, first compute the value $q_0 \in (|R|/M, |R|/M')$ such that $\hat{T}'(q_0) = 0$, which is easily seen to be

$$q_0 = \frac{(1+4\theta)^{1/2} - 1}{2\theta}, \tag{8}$$

where $\theta = (t_a/t_b)(|R|/M')$. Then $\hat{q}^*$ is equal to either $q_0$, $M/|R|$, or $M'/|R|$, depending upon which of the quantities $\hat{T}(q_0)$, $\hat{T}(M/|R|)$, or $\hat{T}(M'/|R|)$ is the smallest.

It is possible to modify the above cost formulas to deal with different systems and applications not covered by our analysis. For example, suppose that the dataset is stored and retrieved in blocks of $b > 1$ items, as is typical for relational tables in commercial RDBMS. Then we can modify our analysis of the phase 1 cost as follows. Let $t_a$ now denote the cost of retrieving a block of items. When $|R| \gg |S|$, so that the probability of an accessed item being rejected during phase 1 sampling is negligible, a Cardenas-type argument shows that the approximate cost of phase 1 is

$$\hat{T}_1(q) = \frac{t_a|R|}{b}\left(1 - \frac{\binom{|R|-b}{N(u_q)}}{\binom{|R|}{N(u_q)}}\right).$$

The foregoing method for choosing $q$ assumes that insertions are the only operations performed on the dataset. The analysis becomes much more complicated in the presence of deletions. Due to space limitations, we leave a detailed study of this topic for future work, and content ourselves here with outlining a simple approach that appears to be promising. We assume that requests arrive every $t_b$ time units as before, and that with probability $p > 1/2$ the request is an insertion and with probability $(1-p)$ the request is a deletion—$p$ can be estimated from observations of the transaction stream $\gamma$. In phase 2, the expected change in the dataset size at each step 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 $1/(2p-1)$. Recall that, in the insertion-only case, the number of Bernoulli trials in phase 2 equals the number of items added to the dataset. Thus, roughly $1/(2p-1)$ times as many steps are required, on average, to finish phase 2 in the presence of deletions. We therefore replace $\hat{T}_2(q)$ by $\hat{T}_2^{\mathrm{del}}(q) = \hat{T}_2(q)/(2p-1)$ and proceed as above.

## 5. EXPERIMENTS

We conducted an experimental study to evaluate the stability and performance of the RP scheme with respect to the various algorithms mentioned in Section 2. Furthermore, we measured the quality of the estimation method for $q^*$ used for resizing. In summary, we found that RP has the following desirable properties:

- RP produces more stable sample sizes than any other algorithm that does not access base data.
- When the fluctuations of the dataset size over time are not too large, RP produces sample sizes that are as stable as those produced by slower algorithms that access the base data.
- RP is as least as fast as any other sampling scheme, and clearly outperforms any sampling scheme that requires access to the base data.

For the resizing algorithm, we found that

- The cost function $\hat{T}(q)$ is almost indistinguishable from the "exact" cost function $T(q)$ that is derived without using the approximation $P\{U = u_q\} = 1$ and without approximating the function $H$. Thus the value $\hat{q}^*$ that is obtained by minimizing $\hat{T}$ is essentially identical to the exact value $q^*$ that is obtained by minimizing $T$; computation of $q^*$ requires numerical optimization algorithms, so that computation of $\hat{q}^*$ is both faster and easier to implement.
- The time needed for resizing has low variance, so that the algorithm has stable performance.

## 5.1 Experimental Setup

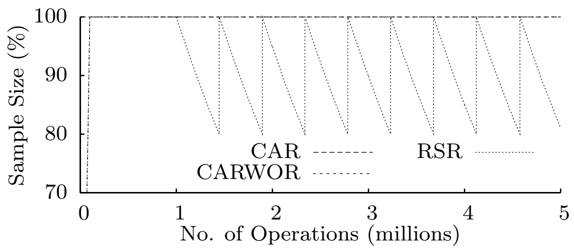
We implemented the new RP algorithm, as well as the CAR, CARWOR, BSP, and RSR schemes, using Java 1.5. We employed an indexed in-memory array to efficiently support the deletion of items—such an index is mandatory for any serious implementation of sampling schemes subject to deletions.

All of the experiments used synthetic data; since our focus is on uniform sampling of unique data items, the actual data values are irrelevant. We ran our experiments on a variety of systems, and measured the number of operations instead of actual processing times in order to facilitate meaningful comparisons. Because the sampling algorithms in this paper can potentially be used in a wide range of application scenarios, our approach has the advantage that the results reported here can be customized to any specific scenario by appropriately costing the various operations. For example, if the base data corresponds to a single relational table, then access to this data can be costed more cheaply than if the base data is, say, a view over a join query. Unless otherwise stated, a reported result represents an average over at least 100 runs.

We assumed that the deletions and insertions are clustered into chunks of $b$ operations, and simulated the sequence of dataset operations by randomly deciding whether the next $b$ operations are insertions or deletions. Our default value was $b = 1$, but we also ran experiments in which we systematically varied the value of $b$ to investigate the effect of different insertion/deletion patterns.

## 5.2 Sample Size

We evaluated the stability of the sample sizes for the various algorithms by executing a randomly generated sequence of 5,000,000 operations while incrementally maintaining a sample with a target size (and upper bound) of 100,000 items. To create a scenario in which the dataset of interest is reasonably large, we restricted the first 1,000,000 operations to be insertions only. We used a lower bound of 80,000 tuples for the RSR algorithm, and an adjustment formula of $q' = 0.8q$ for the purge step of the BSP algorithm. The goal of this experiment is to illustrate the qualitative behavior of the algorithms, and so we did not average over multiple runs. For each algorithm, we plotted the sample size as it evolved over time.[4] The upper part of Fig. 2 displays results for all sampling schemes that access the base data, and the lower part displays results for algorithms that do not require base-data accesses.



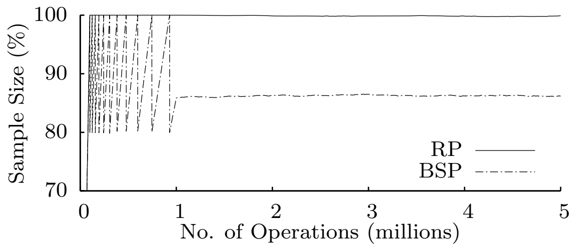


**Figure 2: Evolution of sample size over time**

As can be seen, CAR and CARWOR are optimal, since they are able to maintain the sample at its upper bound. These algorithms, however, need to access the base data. RSR also needs to access the base data, but the sample size is less stable than that of CAR or CARWOR. Comparing algorithms BSP and RP, which do not need to access the base data, we see that the sample sizes produced by RP are almost indistinguishable from those of CAR and CARWOR. Interestingly, the sample sizes produced by BSP fluctuate in roughly the range $[0.8M, M]$ during the startup period (of insertions only), and stabilize at a value strictly less than $M$ thereafter. Indeed, it turns out that BSP does not alter the sampling fraction once the startup period is finished, as long as the sample size does not exceed $M$. If we were to increase the constant in the adjustment formula $q' = cq$ to $c > 0.8$, the sample sizes would be larger, but then the frequency of sample purges would increase, also. Finally, observe that the curves for RSR and BSP have opposite shapes due to the nature of the underlying algorithms.

We next measured the time-average sample size for a range of dataset sizes, providing further insight into the impact of deletions. For each dataset size, we used a sequence of insertions to create both the dataset and the initial sample, and then measured changes in the sample size over time as we inserted and deleted 10,000,000 items at random. The results are shown in Fig. 3. Again, RP performs comparably to CAR and CARWOR, in that it maintains a sample size close to the upper bound $M$. In contrast, the time-average sample sizes for BSP and RSR are smaller than those of the other algorithms. The reason for this behavior is that whereas the reservoir-based approaches continually adjust the sampling fraction, the value of $q$ used by BSP is adjusted at only at certain points of time, i.e., during the purge step. As indicated in Fig. 2, the frequency of such purges is low if the fluctuations in the dataset size over time are not too large (see also the discussion below). Much of the time, therefore, the sample size produced by BSP passively tracks the dataset size, and consequently fluctuates more than algorithms that continually adjust the sample size in an active manner. As a result of this behavior, the sampling rate of BSP is often smaller than the optimal value $M/|R|$. Similarly, the RSR algorithm actively adjusts the sample

[4]We use "time" and "number of operations" synonymously.

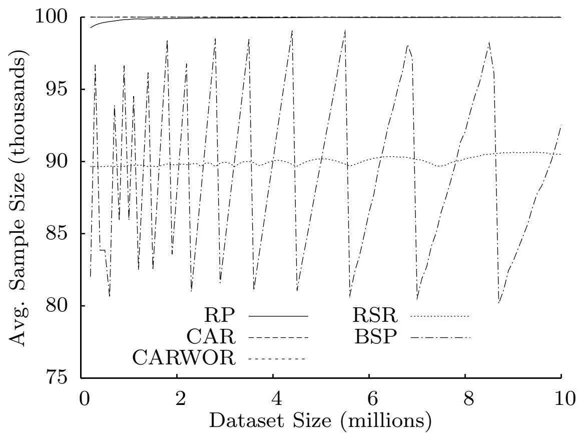


**Figure 3: Dataset Size and avg. sample size**

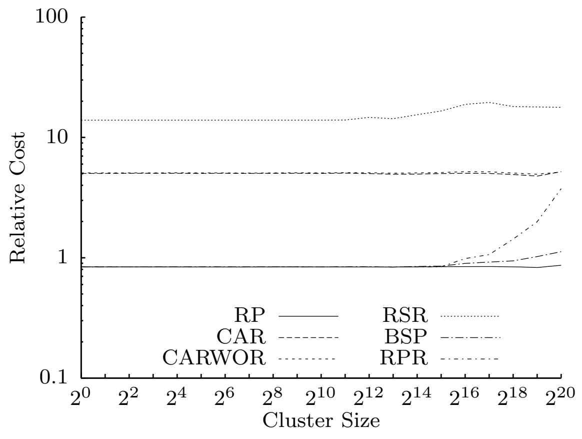


**Figure 5: Cluster size and relative cost**

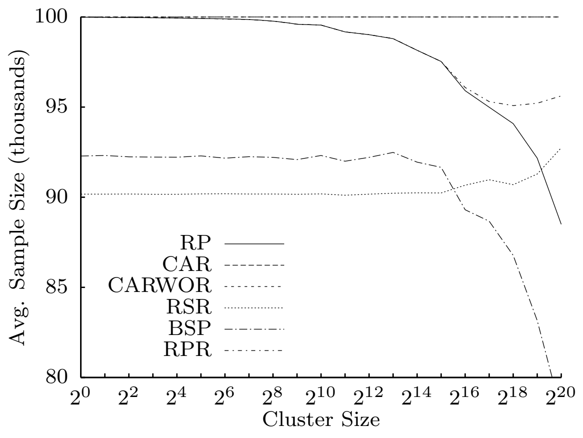


**Figure 4: Cluster size and avg. sample size**

size only periodically, but in contrast to BSP, the average sample size is independent of the dataset size.

The foregoing experiments use a cluster size of $b = 1$, which means that the fluctuations in the dataset size are relatively small. We expect that, when the dataset size fluctuates heavily, so does the sample size when base-data access is disallowed. For example, the sample size produced by RP depends on the number of uncompensated deletions, which in turn is determined as the difference between the current dataset size and the maximum dataset size seen so far. To study this effect experimentally, we varied the magnitude of the fluctuations by varying the cluster size $b$. We started with a dataset consisting of 10,000,000 items and a sample size of 100,000. We then performed $2^{23}$ operations and averaged the sample size after every $b$ operations. The results for different values of $b$ are shown in Fig. 4.

As can be seen, the sample sizes produced by algorithms that access base data are independent of the cluster size, whereas those produced by algorithms that avoid base-data accesses depend on the cluster size. For these latter algorithms, the higher the variance of dataset size, the lower the average sample size. RP performs better than BSP, because RP continuously adjusts the sampling fraction. However, due to high peaks in dataset size, RP as well as BSP may fail to maintain a sufficiently large sample if the cluster size is large with respect to the dataset size. In this extreme case, base-data access is required in order to enlarge the sample. A combination of RP and resizing (RPR) can handle even this situation while minimizing accesses to the base data.[5] Note that RPR guarantees a lower bound on the sample size, whereas BSP and RP do not.

In a final experiment, we measured the overall cost of the various sampling schemes relative to the average sample size produced by them, for various cluster sizes. (Our cost model is described in the next section.) The results are shown in Fig. 5. RSR performs worst since it is expensive and produces a non-optimal sample size; both CAR and CARWOR are more stable and less expensive. Overall, the sampling schemes that do not access base data are clearly superior. Though RP and BSP have approximately the same relative cost, RP produces larger samples. As indicated above, RPR performs comparably to RP when the cluster sizes are reasonably large. The extra cost of RPR come into play when the database size fluctuates very strongly. Indeed, when the cluster size exceeds $2^{20}$, the sample is refilled after almost every deletion block, and RPR reduces to CARWOR.

## 5.3 Performance

To evaluate the relative cost of the sampling algorithms, we ran them using different dataset sizes while counting the number of dataset reads and sample writes. These two factors strongly influence the performance of the algorithms. Again, we created a sequence of 10,000,000 insertions and deletions and averaged the results over various independent runs.

Fig. 6 depicts the number of accesses to base data for the different algorithms. Because it must periodically recompute the entire sample, RSR requires more base-data accesses than any other sampling scheme. Both CAR and CARWOR perform better than RSR, with CARWOR incurring more base-data accesses than CAR due to duplicate removal. All of these algorithms require fewer accesses to a larger dataset than to a smaller one because, for a bounded-size sample, the effective sampling fraction drops with increasing dataset size, so that frequency of deletions

[5]It is not meaningful to extend BSP with resizing since—with high probability—BSP purges the sample immediately, thereby undoing the sample enlargement.

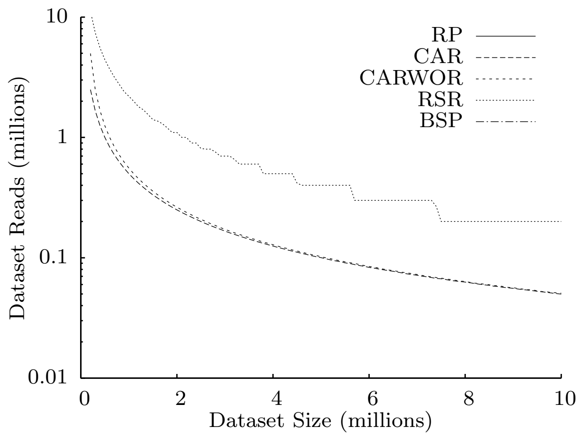


**Figure 6: Number of dataset reads**

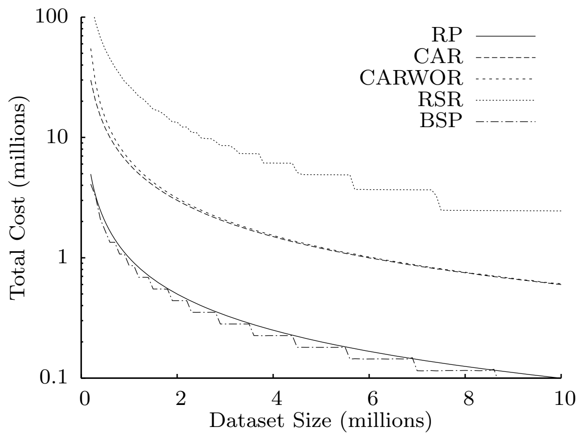


**Figure 8: Combined cost**

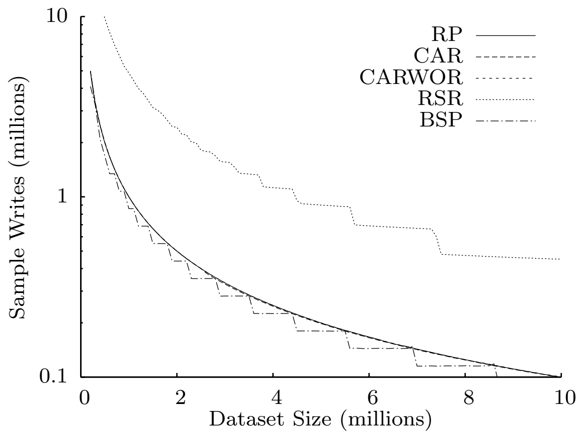


**Figure 7: Number of sample writes**

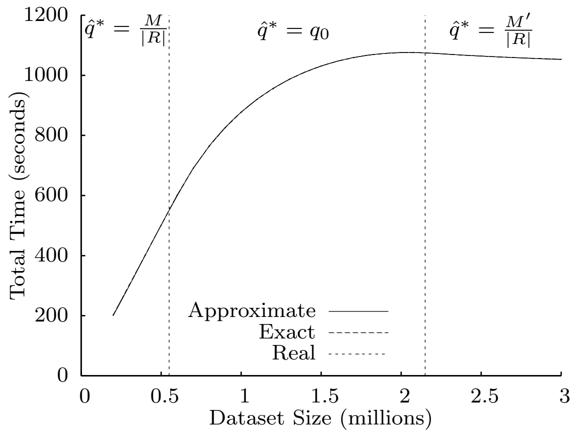


**Figure 9: Cost for resizing**

from the sample drops as well. Note, however, that if large datasets are subject to modifications more often than small ones, then this effect may vanish. Finally, observe that, because neither RP nor BSP ever requires access to the base data, their cost curves are indistinguishable from the $x$-axis.

Fig. 7 shows the number of write accesses to the sample for the different sampling schemes. Again, RSR is the least efficient algorithm, because every recomputation completely flushes the current sample and refills it using base data. The other algorithms perform comparably, with BSP being slightly more efficient.[6]

Fig. 8 shows the combined cost of sample and population accesses, assuming that the latter type of access is ten times as expensive as the former. (In many applications, the relative cost of population accesses might be significantly higher.) Again, BSP is the only competitor to RP, but as already shown, RP yields more stable sample sizes. If the sample size of BSP were increased to compensate for this effect, then RP would be more efficient.

## 5.4 Resizing

We next evaluated the error incurred by approximating the cost function $T(q)$ by the more tractable function $\hat{T}(q)$, assuming that both $t_a$ and $t_b$ are known exactly and that all operations are insertions. We used initial and final sample sizes of $M = 100{,}000$ and $M' = 200{,}000$, respectively, and set $t_a = 10ms$ and $t_b = 1ms$. Besides computing $T(q)$ and $\hat{T}(q)$, we computed the "real" cost based on the above values of $t_a$ and $t_b$ and the actual number of insertions required to complete phase 2 of the resizing algorithm. Fig. 9 depicts the expected cost, its approximation, and the average real cost over 100 independent experiments, along with the variance, all for a range of different dataset sizes. We used the value of $\hat{q}^*$ as the input parameter for the resizing algorithm. As indicated by the figure, the cost of resizing has low variance; indeed, the error bars are too small to be visible. It can also be seen that $T(q)$ and $\hat{T}(q)$ are almost indistinguishable.

The interpretation of the vertical dashed lines in Fig. 9 is as follows. For dataset sizes below the lower threshold, the resizing algorithm sets $q^* = M/|R|$ and initializes the Bernoulli sample as approximately equal to the current sample $S$, thereby avoiding base-data accesses and shifting the sampling work to phase 2. The total resizing cost for dataset sizes in this region is approximately $t_b\big((M'/M) - 1\big)|R|$. As the dataset size increases, the algorithm sets $q^* = q_0$ as in (8), and thereby shifts an increasing share of the work to phase 1. When the dataset size exceeds the upper threshold, the resizing algorithm sets $q^* = M'/|R|$ and simply

[6]The curves for CAR, CARWOR, and RP coincide.

fills up the sample by drawing items from $R \setminus S$, in which case the total cost for resizing is approximately $t_a(M' - M)$. As can be seen, this algorithmic behavior implies that the cost of the resizing algorithm is, to a good approximation, a nondecreasing function of $|R|$ that levels off at the value $t_a(M' - M)$.

## 6. SUMMARY AND CONCLUSIONS

Techniques for incrementally maintaining samples over "datasets"—whether relational tables, views, XML repositories, or other data collections—are crucial for unlocking the full power of database sampling techniques. We have systematically studied algorithms for maintaining a random sample under arbitrary insertions and deletions to the dataset. For stable datasets in which the dataset size does not undergo extreme fluctuations, our new RP algorithm, which generalizes both reservoir sampling and "passive" stream sampling, is the algorithm of choice with respect to speed and sample-size stability. In the presence of severe fluctuations in the dataset size, RP can be combined with resizing or resampling algorithms to achieve acceptable sample sizes while minimizing expensive base-data accesses. For growing datasets, our new resizing algorithm permits the sample size to grow in a controlled manner. For insert-only environments, we have developed effective methods for optimally tuning the algorithm to minimize the time required for resizing. In future work, we plan to extend these tuning techniques to the general setting. We also plan to look at methods for parallelizing our sampling technique and for handling large disk-based samples, perhaps by combining the current algorithms with those in [4], [8] and [17].

## 7. REFERENCES

[1] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *Proc. SODA*, pages 633–634, 2002.

[2] P. Brown, P. Haas, J. Myllymaki, H. Pirahesh, B. Reinwald, and Y. Sismanis. Toward automated large-scale information integration and discovery. In *Data Management in a Connected World*, pages 161–180. Springer, 2005.

[3] P. Brown and P. J. Haas. BHUNT: Automatic discovery of fuzzy algebraic constraints in relational data. In *Proc. VLDB*, pages 668–679, 2003.

[4] P. G. Brown and P. J. Haas. Techniques for warehousing of sample data. In *Proc. ICDE*, 2006.

[5] S. Chaudhuri, R. Motwani, and V. R. Narasayya. On random sampling over joins. In *Proc. ACM SIGMOD*, pages 263–274, 1999.

[6] G. Cormode, S. Muthukrishnan, and I. Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *Proc. VLDB*, pages 25–36, 2005.

[7] G. Frahling, P. Indyk, and C. Sohler. Sampling in dynamic data streams and applications. In *Proc. 21st Symp. Computat. Geom.*, pages 142–149, 2005.

[8] R. Gemulla and W. Lehner. Deferred maintenance of disk-based random samples. In *Proc. EDBT*, pages 423–441, 2006.

[9] P. Gibbons, Y. Matias, and V. Poosala. AQUA project white paper. Technical report, Bell Laboratories, Murray Hill, New Jersey, 1997.

[10] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. ACM SIGMOD*, pages 331–342, 1998.

[11] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. *ACM Trans. Database Syst.*, 27:182–184, 2002.

[12] P. Haas and C. König. A bi-level Bernoulli scheme for database sampling. In *Proc. ACM SIGMOD*, pages 275–286, 2004.

[13] P. J. Haas. Data stream sampling: Basic techniques and results. In *Data Stream Management: Processing High Speed Data Streams*. Springer, 2006. (to appear).

[14] A. Y. Halevy, O. Etzioni, A. Doan, Z. G. Ives, J. Madhavan, L. McDowell, and I. Tatarinov. Join synopses for approximate query answering. In *Proc. CIDR*, 2003.

[15] IBM Corporation. *WebSphere Profile Stage User's Manual*. 2005.

[16] I. F. Ilyas, V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga. CORDS: automatic discovery of correlations and soft functional dependencies. In *Proc. ACM SIGMOD*, pages 647–658, 2004.

[17] C. Jermaine, A. Pol, and S. Arumugam. Online maintenance of very large random samples. In *Proc. ACM SIGMOD*, pages 299–310, 2004.

[18] G. H. John and P. Langley. Static versus dynamic sampling for data mining. In *Proc. KDD*, pages 367–370, 2005.

[19] N. L. Johnson, S. Kotz, and A. W. Kemp. *Discrete Univariate Distributions*. Wiley, 2nd edition, 1992.

[20] J. Kivinen and H. Mannila. The power of sampling in knowledge discovery. In *Proc. ACM PODS*, pages 77–85, 1994.

[21] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, 1st edition, 1969.

[22] U. Leser and F. Naumann. (Almost) hands-off information integration for the life sciences. In *Proc. CIDR*, pages 131–143, 2005.

[23] A. McLeod and D. Bellhouse. A convenient algorithm for drawing a simple random sample. *Applied Statistics*, 32:182–184, 1983.

[24] F. Olken. Random Sampling from Databases. Thesis LBL-32883, Information and Computing Sciences Division, Lawrence Berkeley National Laboratory, 1993.

[25] F. Olken and D. Rotem. Maintenance of Materialized Views of Sampling Queries. In *Proc. ICDE*, 1992.

[26] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proc. ACM SIGMOD*, pages 294–305, 1996.

[27] N. Tatbul, U. Çetintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *Proc. VLDB*, pages 309–320, 2003.

[28] J. Vitter. Random Sampling with a Reservoir. *ACM Trans. Math. Software*, 11(1):37–57, 1985.

[29] J. S. Vitter. Faster Methods for Random Sampling. *Commun. ACM*, 27(7):703–718, 1984.