TECHNISCHE UNIVERSITÄT DRESDEN

Sampling Algorithms for Evolving Datasets

Kurzfassung der Dissertation

zur Erlangung des akademischen Grades
Doktor rerum naturalium (Dr. rer. nat.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
Dipl.-Inf. Rainer Gemulla
geboren am 28. April 1980 in Sondershausen

Dresden im August 2008

Gutachter: Prof. Dr.-Ing. Wolfgang Lehner
Technische Universität Dresden
Fakultät Informatik, Institut für Systemarchitektur
Lehrstuhl für Datenbanken
01062 Dresden

Dr. Peter J. Haas
IBM Almaden Research Center, K55/B1
650 Harry Road, San Jose, CA 95120-6099
USA

Prof. Dr.-Ing. Dr. h.c. Theo Härder
Technische Universität Kaiserslautern
Fachbereich Informatik
AG Datenbanken und Informationssysteme
67653 Kaiserslautern

Sampling Algorithms for Evolving Datasets (Extended Abstract)

Rainer Gemulla

August 27, 2008

Perhaps the most flexible synopsis of a database is a uniform 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. Most of the existing work on database sampling focuses on how to create or exploit a random sample of a static database, that is, a database that does not change over time. The assumption of a static database severely limits the applicability of these techniques in practice, where data is often not static but continuously evolving. In the thesis,[1] 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, updates, and deletions. We also discuss algorithms for resizing random samples upwards and downwards, derive novel estimators for certain population parameters, and provide methods for combining two or more random samples into a sample of the combined datasets. Our algorithms can potentially be leveraged to extend the applicability of many previous database sampling techniques to the class of evolving datasets.

1 Introduction

Recent studies conducted by IDC have revealed that the 2007 "digital universe" comprises about 45 gigabytes of data per person on the planet [38, 39]. Looking only at the data stored in large-scale data warehouses, current estimates indicate that the size of the world's largest warehouse triples about every two years, thereby even exceeding Moore's law [52]. To analyze this enormous amount of data, random sampling techniques have proven to be an invaluable tool. They have numerous applications in the context of data management, including query optimization [20, 22, 35, 45, 46], load balancing [16, 17], approximate query processing [1–3, 8, 11–13, 23, 34, 37, 41–43, 48, 53], statistics estimation [9, 10, 31, 49], data mining [5–7, 14, 29, 32, 40, 44],

[1] This paper is an overview of Ph.D. thesis [24]. For brevity, we refer to [24] as "the thesis".

and data stream processing [3, 33]. In these applications, random sampling techniques are exploited in two fundamentally different ways: (i) they help compute an *exact* query result *efficiently* and (ii) they provide means to *approximate* the query result. In both cases, the use of sampling may significantly reduce the cost of query processing.

For an example of (i), consider the problem of deriving an "execution plan" for a query expressed in a declarative language such as SQL. There usually exist several alternative plans that all produce the same result, but they can differ in their efficiency by several orders of magnitude; we clearly want to pick the plan that is most efficient. In the case of SQL, finding the optimal plan includes (but is not limited to) decisions on the indexes to use, on the order in which to apply predicates, on the order in which to process joins, and on the type of sort/join/group-by algorithm to use. Query optimizers make this decision based on estimates of the size of intermediate results. Virtually all major database vendors—including IBM, Microsoft, Oracle, Sybase, and Teradata—use random sampling to compute online and/or precompute offline statistics that can be leveraged for query size estimation. This is because a small random sample of the data often provides sufficient information to separate efficient and inefficient plans.

Perhaps the most prevalent example of (ii) is approximate query processing. The main idea behind this processing model is that the computational cost of query processing can be reduced when the underlying application does not require exact results but only a highly-accurate estimate thereof. For instance, query results visualized in a pie chart may not be required to be exact up to the last digit. Likewise, exploratory "data browsing"—carried out in order to find out which parts of a dataset contain interesting information—greatly benefits from fast approximate query answers. It is not surprising that random sampling is one of the major techniques in approximate query processing. There exists a large body of work on how to compute and exploit random samples; results obtained from a random sample can be enriched with information about their precision and, if desired, progressively refined; and sampling scales well with the size of the underlying data. Recognizing the importance of random sampling for approximate query processing, the SQL standardization committee included basic sampling clauses into the SQL/Foundation:2003 standard; these clauses are already implemented in most commercial database systems.

Throughout the thesis, we focus entirely on uniform random sampling, in which all samples of the same size are equally likely. Uniform sampling is the most basic of the available sampling designs; it is ubiquitous in applications. In fact, most statistical estimators—as well as the confidence-bound formulas for these estimators—assume an underlying uniform sample. Moreover, uniformity is a must if it is unknown in advance how the sample will be used; this situation occurs frequently in database sampling. 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 general, there are two alternative approaches to sample computation. First, *query sampling schemes* obtain the sample on the fly as needed. In the setting of commercial RDBMS, Haas and König [34] have shown that there is a trade-off

between the uniformity of a query sampling scheme and the cost of obtaining the sample. Even if some degree of non-uniformity is acceptable, query sampling can still be too expensive. In contrast, *materialized sampling schemes*—which initially materialize a sample from the underlying dataset and then incrementally maintain it over time—amortize the cost of obtaining the sample over its subsequent usages. We can "invest" in sophisticated sampling designs well-suited for our specific application, even if such a sample were too costly to obtain at query time. Another distinctive advantage of materialized sampling is that access to the underlying data is not required in the estimation process. The more expensive the access to the actual data, the more important this property gets. In fact, base data accesses may even be infeasible in applications in which, for instance, the underlying dataset is not materialized or resides at a remote location.

The main deficiency of existing techniques for maintaining a materialized sample is that they either are restricted to the class of append-only datasets, in which data once inserted is never changed or removed, and/or they require expensive base-data accesses. The thesis contributes novel *maintenance* algorithms for the general class of evolving datasets, in which the data is subject to insertion, update and deletion transactions. Our algorithms are truly *incremental* in that they maintain the sample based solely on the stream of transactions and without ever accessing the underlying dataset. We also show how our samples can be exploited to derive novel *estimators* for counts, sums, averages, and the number of distinct items in the underlying dataset. In addition to sample maintenance, we discuss methods that greatly improve the flexibility of random sampling from a system's point of view. More specifically, we initiate the study of algorithms that *resize* a random sample upwards or downwards. Our resizing algorithms can be exploited to dynamically control the size of the sample when the dataset grows or shrinks; they facilitate resource management and help avoid under- or oversized samples. Furthermore, in large-scale databases with data being distributed across several remote locations, it is usually infeasible to reconstruct the entire dataset for the purpose of sampling. To address this problem, we provide efficient algorithms that directly *combine* the local samples maintained at each location into a sample of the global dataset. We also consider a more general problem, where the global dataset is defined as an arbitrary set or multiset expression involving the local datasets, and provide efficient solutions based on hashing.

The remainder of this paper is structured as follows: In section 2, we review the basic techniques that underlie all database sampling schemes and discuss different areas of database sampling. Sections 3–6 highlight our contributions to each of those areas. Section 7 concludes this paper.

2 Uniform Sampling

In this section, we describe the sampling problem more precisely and give an overview of the basic sampling schemes that underlie all of the available database sampling schemes. We call a sampling scheme *uniform* if the probability $p_R(S)$ that the scheme

produces sample S when applied to dataset R satisfies $p_R(S) = p_R(S')$ 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 our attention to sampling without replacement; in general, a without-replacement sample contains more statistical information about the dataset than a with-replacement sample of the same size [50].

2.1 Notation

We model the base dataset R as a finite subset of a (possibly infinite) set $\mathscr{R} = \{ r_1, r_2, \dots \}$ of unique, distinguishable *items*, and the sample S is, in turn, a subset of R. For example, $\mathscr{R}$ might correspond to a finite set of IP addresses, an infinite sequence of unique text or XML documents, or perhaps a set of relational tuples. Without loss of generality, we assume throughout that the dataset R is initially empty, and evolves over time as items are inserted and deleted. Items can be inserted more than once so that R is a multiset, and items that are deleted may be subsequently re-inserted. Thus, we consider an infinite sequence of transactions $\gamma = (\gamma_1, \gamma_2, \dots)$, where each transaction γ_i is either of the form $+r_k$, which corresponds to the insertion of (a single copy of) item r_k into R, or of the form $-r_k$, which corresponds to the deletion of (a single copy of) item r_k from R. We restrict attention to "feasible" sequences such that $\gamma_i = -r_k$ only if item r_k is in the dataset just prior to the processing of the ith transaction.[2] Our goal is to ensure that, after each transaction is processed, S is a uniform sample from R. As is usual in practice, we assume throughout that the sequence γ of insertions and deletions to the data is oblivious to the behavior of the sampling algorithm.

2.2 Survey Sampling Methods

We first discuss three classic sampling schemes: Bernoulli sampling, reservoir sampling, and min-wise sampling. Each of these schemes has its roots in computer-assisted survey sampling and supports only insertion transactions. The applicability of the schemes to database sampling under general transactions is discussed in the subsequent sections, where we also outline the contributions of the thesis in more detail.

Bernoulli sampling, BERN(q) In the Bernoulli sampling scheme with sampling rate 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$(|R|, q)$ distribution, so that $\Pr[\,|S| = k\,] = \binom{|R|}{k} q^k (1-q)^{|R|-k}$ for $k = 0, 1, \dots, |R|$ and $E[|S|] = q|R|$. Although the sample size is random, samples of the same size are equally likely, and the scheme is indeed uniform, as defined previously. The main advantages of Bernoulli sampling are simplicity and ease of

[2] We focus on insertion and deletion transactions; updates can be modeled by a deletion followed by an insertion.

parallelization. The sample size is unbounded, though sharply concentrated around the expected value of $q|R|$.

Reservoir sampling, RS(M) This uniform scheme maintains a random sample of fixed size M, given a sequence of insertions. The procedure, as described in [47], 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. The main advantage of reservoir sampling is its fixed sample size, which facilitates memory management and bounds the runtime cost of exploiting the sample.

Min-wise sampling, MIN(M) In min-wise sampling, a random "tag" drawn independently from the uniform$(0, 1)$ distribution is associated with each arriving item. The sample consists of the items with the M smallest tags seen thus far. Uniformity follows by symmetry: every size-M subset of R has the same chance of having the smallest tags. Compared to reservoir sampling, min-wise sampling has the disadvantage that its sample footprint is larger because additional space is required to store the tags. It also has a higher CPU cost. Min-wise sampling has the advantage, however, that the number of items in R does not have to be known when executing a sampling step.

2.3 Database Sampling Methods

As mentioned previously, one of the main challenges in database sampling is that the datasets of interest are evolving; they are subject to insertion, update and deletion transactions. Survey sampling methods cannot be used directly because they support neither update nor deletion transactions. There has been a significant amount of research on maintaining database samples [3, 8, 13, 14, 18, 19, 29–31, 33, 36, 48, 51], and many of the existing methods are based upon the survey sampling methods described above. Database sampling methods can be roughly classified based on the type of the underlying dataset and the type of the desired sample:

- *Set sampling* (section 3). At any time, dataset R is a real set, that is, R does not contain duplicates. This is a common scenario for sampling from relational tables, where primary key constraints prevent the insertion of duplicates.

- *Multiset sampling* (section 4). Dataset R is a multiset, that is, R may contain duplicates. As a consequence, sample S is also a multiset. This is common for sampling from a view over several relational tables, where the primary keys are omitted.

- *Distinct-item sampling* (section 5). Dataset R is a multiset but the sample is drawn from the distinct items in R. Sample S is a set. This scenario occurs when sampling from a view with duplicate-eliminating projections; it also arises in various network sampling problems.

Table 1: Existing sampling schemes (✓) and novel sampling schemes (name)

From	Size/Space	Insertions	Updates	Deletions
Sets	Bernoulli	✓	✓	✓
	bounded	✓	✓	RP(M)
Multisets	Bernoulli	✓	ABERN(q)	ABERN(q)
	bounded	✓	?	?
Distinct items	Bernoulli	✓	✓	✓, ABERND(q)
	bounded	✓	✓	✓, AMIND(M)
Time-based windows	Bernoulli	✓	n/a	n/a
	bounded	BPS(M), MBS	n/a	n/a

- *Data stream sampling* (section 6). Dataset R is defined as a sliding window over a data stream. There are no updates or deletions, but items expire after a certain amount of time has passed since their arrival. Data stream sampling methods are used to sample from the recent items in the stream, when older items are considered outdated and irrelevant for analysis.

The derivation of efficient methods for each of the four scenarios requires unique innovation. Note that, in general, multiset sampling schemes and distinct-item sampling schemes can be used to implement set sampling, although this approach is usually less efficient than native set sampling.

An overview of the sampling schemes proposed in the thesis is given in table 1. The first two columns refer to the type of sampling and the sample size distribution. The remaining columns refer to the different types of transactions. A check mark (✓) indicates that there are previously known schemes that can maintain a sample incrementally under the respective transaction type. Novel schemes presented in the thesis are indicated by their abbreviated names; each listed scheme also supports the transactions corresponding to the columns to the left of it. A question mark (?) means that no known algorithms exist and "n/a" means not applicable.

In table 1, we distinguish *bounded sampling schemes*, in which the sample size is bounded from above, and *Bernoulli sampling schemes*, in which the sample size is binomially distributed. Bounded sampling schemes are the method of choice for *stable datasets*, whose size (but not necessarily composition) remains roughly constant over time. The main challenge is to maximize sampling *efficiency*, i.e., to keep the sample size as close to the upper bound as possible. In contrast, Bernoulli sampling methods are well-suited for *growing datasets*, in which insertions occur more frequently than deletions over the long run. The expected sample size is proportional to the dataset size so that the sample grows and shrinks with the dataset. Additional control over the sample size distribution is provided by appropriate resizing methods that increase or decrease the size of the sample; we provide such methods for all our schemes.

In the rest of this paper, we outline the contribution of the thesis to the four areas of database sampling mentioned above. Each of the subsequent sections is structured into the 4 parts: existing techniques, novel techniques, resizing techniques, and merging techniques.

3 Set Sampling

Existing techniques It is straightforward to modify BERN(q) sampling on sets to handle deletions. The algorithm remains unchanged for insertions; items deleted from R are also removed from the sample, if present. Thus a deletion operation of the form $-r$ "annihilates" item r; it is as if item r were never inserted into R. This modified Bernoulli sampling scheme, denoted MBERN(q), has zero overhead, does not require access to the base data and is simple to implement. In applications where the sample size variability and unboundedness of Bernoulli sampling are acceptable, the scheme is clearly the method of choice.

Many different techniques have been proposed in order to extend reservoir sampling with support for deletion transactions. The basic problem is that the deletion of an item that is present in the sample leads to a reduction of the sample size and thus to a reduction of the sampling efficiency. We cannot simply continue sampling with the smaller sampling size because, when doing so, the sample size would systematically converge to 0. To overcome this problem, [31] proposes to recompute the sample from scratch whenever its size has fallen below a prespecified lower bound. Similarly, [48] proposes to replace deleted items by random items drawn uniformly from the underlying dataset. Both approaches are expensive because they rely on frequent accesses to the underlying dataset. A different approach is taken in [51], where deletion transactions are essentially treated as "updates that tag an item as deleted." The scheme works well when deletions are infrequent but the sample size converges to 0 for stable datasets so that recomputation is required from time to time. Thus, all known methods require expensive base data access to deal with deletion transactions; they differ with respect to the number of these accesses and the sample-size stability.

Novel techniques An ideal bounded-size scheme for stable datasets would not require any base data accesses while at the same time providing a fixed sample size. Unfortunately, such a scheme does not exist: every sample deletion either leads to a decrease of the sample size or, if immediately compensated, to a base data access. In the thesis, we introduce a novel bounded-size scheme for stable datasets that comes close to the ideal scheme. Our scheme is called *random pairing* [26, 28] and denoted as RP(M). The key idea of random pairing is to make use of newly inserted items to compensate for prior deletions. Conceptually, the scheme pairs each incoming insertion transaction with a previous deletion transaction, the "partner" deletion, and then compensates the partner. Thus, each deletion is compensated by precisely one insertion. We show in the thesis that in order to execute the pairing step, it suffices to maintain two counters: one for the number of "bad" uncompensated

deletions, which have been in the sample at the time of their deletion, and one for the number of "good" uncompensated deletions, which have not been in the sample. Furthermore, we prove the correctness of the algorithm and analyze its statistical properties. In contrast to all previously known schemes, random pairing does not require access to the base data, even if the transaction sequence contains deletions. Our experiments indicate that, when the fluctuations of the dataset size are not too extreme, random pairing is the algorithm of choice with respect to speed and sample-size stability.

Resizing techniques We initiate the study of algorithms for periodically "resizing" a bounded-size random sample upwards, proving that any such algorithm cannot avoid accessing the base data. Such methods are of interest for growing datasets because they can be exploited to grow the sample in a controlled manner, whenever needed. Prior to our current work, the only proposed approach to the resizing problem was to naively recompute the sample from scratch. We provide a novel resizing algorithm called RPRES that partially enlarges the sample using the base data, and subsequently completes the resizing using only the stream of insertion, update and deletion transactions. Especially when access to the base data is expensive and transactions are frequent, the resizing cost can be significantly reduced relative to the naive approach. We also give a subsampling algorithm termed RPSUB that can be used to reduce the size of the sample; the algorithm is useful to handle potential memory bottlenecks and to undo prior sample enlargements.

Merging techniques Finally, we discuss algorithms for merging two or more uniform samples. Merging algorithms are particularly useful when the dataset is partitioned over several nodes. In this case, sample merging can be used to obtain a sample of the complete dataset from local samples maintained at each node, thereby facilitating efficient parallel sampling. Our new RPMERGE algorithm extends the MERGE algorithm of [8]—which was developed for an insertion-only environment—to effectively deal with deletions. We show analytically that RPMERGE produces larger sample sizes than MERGE in expectation; the merged sample thus contains more information about the complete dataset. Additionally, when RPMERGE is used, the merged sample can be maintained incrementally using our random pairing algorithm.

4 Multiset Sampling

Existing techniques Relatively little is known about sampling from evolving multisets. The survey sampling schemes BERN(q), RS(M), and MIN(M) can all be used to maintain a sample of an insertion-only multiset. This is because the inclusion/exclusion decisions are independent of the value of the inserted item. To derive the multiset versions of the algorithms, simply treat the sample S as a multiset. When the transaction sequence contains updates or deletions, however, maintenance becomes much harder because both the dataset R and the sample S may contain

multiple copies of the updated or deleted item. Since update and deletion transactions refer to only a single one of these copies, it is not immediately clear how to proceed.

We subsequently assume that the sample is stored in compressed form, as proposed in [29]. In the compressed representation, each element of the sample comprises a pair (r, f_r), where $f_r > 0$ denotes the frequency of item r in the sample. The only previously known scheme that supports deletions is a multiset version of the MBERN(q) scheme outlined above [27]. The scheme processes an insertion transaction $+r$ as before but takes care of sample compression: in case an item r is accepted into the sample, the counter f_r is increased by one or, when r has not been sampled previously, a pair $(r, 1)$ is added to the sample. A deletion of the form $-r$ is ignored when $r \notin S$. Otherwise, the scheme decreases the counter f_r with probability f_r/N_r, where N_r is the frequency of r in the base dataset R. Laxly speaking, when f_r of the N_r copies of item r are present in the sample, the "deleted copy" is one of the copies in the sample with probability f_r/N_r. Unfortunately, the scheme is not incremental because it requires knowledge of the quantity N_r, which has to be obtained from the underlying data.

Novel techniques We propose a sampling scheme called *augmented Bernoulli sampling*, ABERN(q), that is able to maintain a Bernoulli sample of an evolving multiset without ever accessing base data [27]. Our method augments each distinct sample item with a "tracking counter," originally introduced in Gibbons and Matias [29] for the purpose of estimating the population frequencies of "hot" items. These counters "track" the net number of insertions of each sample item since its initial acceptance into the sample. Thus, the value of the tracking counter of a sample item r is at least f_r and at most N_r; it can be anything in between. We show that knowledge of the tracking counters is sufficient to maintain the sample incrementally. In contrast to previous schemes, neither base data accesses nor knowledge of N_r are required. The tracking counters not only facilitate incremental maintenance, but they can also be exploited to obtain unbiased estimators of population frequencies, sums, and averages, where these estimators have lower variance than the usual estimators based on an ordinary Bernoulli sample. Furthermore, we show how to estimate the number of distinct items in the multiset in an unbiased manner. Our distinct-item estimator is based on the observation that a distinct-item Bernoulli sample can be extracted from our ABERN(q) sample; this is not possible with ordinary Bernoulli samples. We derive the standard error for each of our estimators, and provide formulas for estimating these standard errors.

Resizing techniques As for set sampling, we discuss the problem of dynamically changing the sampling rate q of our augmented samples. We argue that increasing the value of q so as to enlarge the sample size requires a scan of almost the entire dataset. Fortunately, this situation occurs rarely in practice because ABERN(q) samples already grow linearly with the dataset size. A more interesting problem is to reduce the value of q in order to reduce the sample size and to avoid oversized samples. We give a subsampling algorithm that performs such a reduction of q without accessing

the base data; the resulting sample can be incrementally maintained. Note that subsampling cannot be used to enforce strict bounds on the sample size because such an approach would lead to non-uniform samples; see the thesis for details. However, we show how subsampling can be used to provide tight probabilistic bounds, which may suffice in practice.

Merging techniques As a negative result for merging operations, we show that, in the general case of non-disjoint parent datasets, it is impossible to merge augmented samples to obtain a new augmented sample from the union of the corresponding parent datasets. However, the augmented samples can still be used to obtain a plain, non-maintainable Bernoulli sample from this union.

5 Distinct-Item Sampling

Existing techniques Distinct-item sampling schemes sample uniformly from $D(R)$, the set of distinct items in R. All previously known schemes are based on hashing, that is, they make use of a set of hash functions $h_1, \ldots, h_k$ for some $k \geq 1$. The sampling schemes are deterministic in the sense that they produce the same result when run repeatedly on the same transaction sequence using the same set of hash functions. The "randomness" of the sample thus depends only on the degree of randomness in the hash functions. In the following, we assume that the hash functions being used are truly random in the real $(0, 1)$ interval; see the thesis for a discussion of the suitability of available hash functions.

The BERN(q) scheme can be adapted to sample uniformly from the distinct items of an insertion-only dataset [15, 18, 19, 30, 36]. The idea is simple: the decision of whether or not to include an item is based on its hash value: an arriving item is accepted into the sample if and only if $h(r) < q$, that is, its hash value is less than q. Since $h(r)$ is uniform in $(0, 1)$, we have $\Pr[\, h(r) < q \,] = q$ as desired. The scheme treats the sample as a set. Since repeated insertions of the same item lead to the same random hash value, each item is either accepted at its first insertion or it is never going to be accepted at all. It follows directly that the probability that an item is sampled does not depend on its frequency in R. In order to support deletion transactions, it is necessary to store the frequency N_r of each sampled item together with the sample. In contrast to multiset sampling, we are able to maintain N_r for the items in the sample, so that the scheme is fully incremental.

The same trick cannot be used with reservoir sampling, but it is possible to apply it to min-wise sampling. The idea is to run min-wise sampling as before but to replace the random tags by the hash value of the arriving item. For this reason, the algorithm is referred to as *min-hash sampling* [5]; it supports only insertion transactions. Another bounded-size scheme called *dynamic inverse sampling* is given in [13, 21]. The scheme is incremental but has significant time and space overhead.

Novel techniques As mentioned previously, we propose a distinct-item scheme based on our ABERN(q) algorithm for sampling multisets. The new scheme is called

ABERND(q); it constitutes the only distinct-item scheme that does not make use of hashing. However, the size of the underlying ABERN(q) sample—from which the ABERND(q) sample is extracted—depends on the size of the entire dataset instead of just the number of distinct items in it. When the dataset contains many frequent items and a multiset sample is not required, the distinct-item Bernoulli sampling algorithm described above is usually a better choice.

In addition to ABERND(q), we propose a simple extension to min-hash sampling that augments the sample with frequency counters. Our extension is called *augmented min-hash sampling*, denoted as AMIND(M), and adds support for a limited number of updates and deletions. AMIND(M) is similar to the "reservoir sampling with tagging scheme" in [51], that is, deleted items are occasionally retained (but not reported) in the sample in order to ensure uniformity. In our analysis, we show that only deletions of the last copy of a distinct item may have a negative impact on the sample size, and that the sample stays reasonably large when the fraction of last-copy deletions is not too high. We also consider the problem of estimating the number of distinct items in the dataset from plain MIND(M) and our AMIND(M) samples [4].[3] We present several estimators for this problem and analyze their theoretical properties. It turns out that our estimators are unbiased and have low variance; they can even be used to estimate the number of distinct items that satisfy a given predicate.

Resizing techniques Unfortunately, it appears that every algorithm that resizes an AMIND(M) sample upwards must scan almost the entire base data. We give a slightly more efficient algorithm than naive recomputation from scratch. Reducing the sample size is significantly easier and can be done without accessing base data; the resulting sample can be incrementally maintained.

Merging techniques and other combinations One of the most interesting properties of AMIND(M) samples is that they can be combined to obtain samples of arbitrary unions, intersections and differences of their underlying datasets. In fact, we show that AMIND(M) samples are closed under these operations. Therefore, AMIND(M) samples are much more powerful in this respect than the set and multiset sampling schemes discussed earlier, which support only merging operations. As might be expected, the size of the resulting sample depends on the selectivity of the expression used to combine the samples, so that our techniques can only be used when the result of the expression is not too small.

6 Data Stream Sampling

For data stream sampling, we consider algorithms that sample from a time-based sliding window defined over a data stream. Such a window consists of all the items that arrived in the last Δ time units and is frequently used to restrict analysis to a

[3] Maintenance of the exact number of distinct items requires space linear to $|D(R)|$, which is infeasible in practice.

recent time horizon. Note that the *window length* Δ is a fixed parameter specified in advance, while the *window size*—i.e., the number of items in the window—varies over time and is generally unbounded. Data stream algorithms usually work under hard time and space constraints; we assume that it is infeasible to store and process the entire window so that sampling methods are needed.

Existing techniques The modified Bernoulli sampling scheme of section 3 can directly be used on sliding windows [33]. For bounded-size sampling, [3] proposes a scheme called *priority sampling*, denoted as PS(M), which makes use of an idea similar to min-wise sampling. The basic scheme, denoted PS(1), maintains a single-item sample; larger with-replacement samples are obtained by running M independent PS(1) samplers in parallel. The PS(1) scheme assigns a random "priority" chosen uniformly from the unit interval $(0, 1)$ to each arriving item. The idea is to report at any time the item with the highest priority in the window as the sample item. Uniformity then follows because each item has the same probability of having the highest priority. To maintain the highest-priority item incrementally, PS(1) makes use of a chain of "replacement items" that replace the current sample item when it expires. It can be shown that the length of the chain averages to $O(\log N)$, where N is the window size. Since the sample size is 1, the scheme has a multiplicative space overhead of $O(\log N)$. Since N cannot be bounded, neither can the space consumption of PS(1).

Novel techniques We prove as a negative result that no bounded-space uniform sampling scheme over a time-based sliding window can guarantee a minimum sample size. As a byproduct, we infer that PS(M) is optimal for fixed-size sampling in terms of space consumption. Nevertheless, in spite of being optimal, PS(M) has a high space overhead that leads to low space efficiency. To sample in bounded space, we develop a related scheme called *bounded priority sampling*, BPS(M), which can be seen as a modification of priority sampling, although the underlying idea is different [25]. BPS(M) cannot provide strict sample size guarantees but it is able to provide strong probabilistic ones. In fact, our experiments indicate that—in addition to enforcing space bounds—BPS(M) has higher space efficiency than PS(M). Moreover, we show how to sample without replacement and how to estimate the window size directly from the sample by leveraging our estimation techniques for distinct-item sampling.

We also propose a stratified sampling scheme for time-based sliding windows. In stratified sampling, the dataset is divided into a set of disjoint strata and a sample is taken from each of these strata (e.g., using one of the schemes proposed in the thesis). Usually, stratified sampling is used to improve the quality of the estimates derived from the sample. The situation is different for data stream sampling because we can exploit stratification to facilitate efficient sample maintenance. In fact, our *merge-based stratification* algorithm divides the window into strata of approximately equal size; it then maintains a uniform sample of each stratum. The algorithm merges adjacent strata from time to time; the main challenge is to decide when and

which strata to merge. In our solution, we treat the problem as an optimization problem and give a dynamic programming algorithm to determine the optimum stratum boundaries. The resulting algorithm, termed MBS, has even higher space efficiency than BPS(M). The downside is that stratified samples cannot be used with all applications so that the advantages of MBS cannot always be exploited.

7 Conclusion

Due to its wide range of applications, random sampling has been and continues to be an important research area in data management. A quick look at the bibliography of the thesis reveals that roughly 80 sampling-related papers appeared in the last decade, counting only major database conferences and journals, and roughly 35 of them appeared in the last 4 years. Many of the sampling techniques proposed in these papers have been developed for static datasets, in which a sample once computed remains valid for all times. In practice, however, datasets evolve and any changes to the data have to be appropriately reflected in the sample to maintain its statistical validity. In the thesis, we address this problem and provide efficient methods for sample maintenance. Our algorithms can potentially be leveraged to extend the applicability of many previous techniques to the class of evolving datasets.

More specifically, we consider the problem of maintaining a uniform random sample of an evolving dataset; such samples are a building block of more sophisticated techniques. We propose novel maintenance algorithms including *random pairing* (for sets), *augmented Bernoulli sampling* (for multisets), *augmented min-hash sampling* (for distinct items), *bounded priority sampling* (for data streams), and *merge-based stratified sampling* (also for data streams). The key property of all our algorithms is that they are incremental, that is, they completely avoid any expensive accesses to the underlying dataset. Instead, sampling decisions are based solely on the stream of insertion, update, and deletion transactions applied to the data. We propose novel estimators for counts, averages, sums, ratios, and the number of distinct items. Our estimators exploit the maintenance information stored along with the samples and they have lower estimation errors than known estimators, which do not exploit this information. In addition to our maintenance schemes, we present algorithms for resizing a random sample and for combining several samples into a single one. These techniques enhance the flexibility of random sampling from a system's point of view.

References

[1] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridha Ramaswamy. Join synopses for approximate query answering. In *Proc. of the 1999 ACM SIGMOD Intl. Conf. on Management of Data*, pages 275–286, 1999.

[2] Swarup Acharya, Phillip B. Gibbons, and Viswanath Poosala. Congressional samples for approximate answering of group-by queries. In *Proc. of the 2000*

ACM SIGMOD Intl. Conf. on Management of Data, pages 487–498, 2000. URL http://citeseer.nj.nec.com/acharya99congressional.html.

[3] Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling from a moving window over streaming data. In *Proc. of the 2002 Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 633–634, 2002.

[4] 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.

[5] Andrei Z. Broder. On the resemblance and containment of documents. In *Proc. of the 1997 Compression and Complexity of Sequences*, pages 21–29, 1997.

[6] Hervé Brönnimann, Bin Chen, Manoranjan Dash, Peter Haas, Yi Qiao, and Peter Scheuermann. Efficient data-reduction methods for on-line association rule discovery. In Krishnamoorthy Sivakumar Hillol Kargupta, Anupam Joshi and Yelena Yesha, editors, *Data Mining: Next Generation Challenges and Future Directions*. AAAI Press, 2004.

[7] Paul Brown and Peter J. Haas. Bhunt: Automatic discovery of fuzzy algebraic constraints in relational data. In *Proc. of the 2003 Intl. Conf. on Very Large Data Bases*, pages 668–679, 2003.

[8] 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.

[9] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: how much is enough? In *Proc. of the 1998 ACM SIGMOD Intl. Conf. on Management of Data*, pages 436–447, 1998.

[10] S. Chaudhuri, G. Das, and U. Srivastava. Effective use of block-level sampling in statistics estimation. In *Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of Data*, pages 287–298, 2004.

[11] Surajit Chaudhuri, Gautam Das, Mayur Datar, and Rajeev Motwaniand Vivek R. Narasayya. Overcoming limitations of sampling for aggregation queries. In *Proc. of the 2001 Intl. Conf. on Data Engineering*, pages 534–544, 2001.

[12] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems*, 32(2):9, 2007. ISSN 0362-5915.

[13] 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.

[14] Mayur Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. In *Proc. of the 2002 Annual European Symp. on Algorithms*, pages 323–334. Springer-Verlag, 2002.

[15] Dorothy E. Denning. Secure statistical databases with random sample queries. *ACM Transactions on Database Systems*, 5(3):291–315, 1980. ISSN 0362-5915.

[16] David J. DeWitt, Jeffrey F. Naughton, and Donovan A. Schneider. Parallel sorting on a shared-nothing architecture using probabilistic splitting. In *Proc. of the 1991 Intl. Conf. Parallel and Distributed Information Systems*, pages 280–291, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[17] David J. DeWitt, Jeffrey F. Naughton, Donovan A. Schneider, and S. Seshadri. Practical skew handling in parallel joins. In *Proc. of the 1992 Intl. Conf. on Very Large Data Bases*, pages 27–40, 1992.

[18] N. G. Duffield and Matthias Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Transactions on Networking*, 9(3):280–292, 2001. ISSN 1063-6692.

[19] Nick Duffield, Carsten Lund, and Mikkel Thorup. Charging from sampled network usage. In *Proc. of the 2001 ACM SIGCOMM Workshop on Internet Measurement*, pages 245–256, New York, NY, USA, 2001. ACM.

[20] Cristian Estan and Jeffrey F. Naughton. End-biased samples for join cardinality estimation. In *Proc. of the 2006 Intl. Conf. on Data Engineering*, page 20, 2006.

[21] Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. In *Proc. of the 2005 Annual ACM Symp. on Computational Geometry*, pages 142–149, 2005.

[22] Sumit Ganguly, Phillip B. Gibbons, Yossi Matias, and Abraham Silberschatz. Bifocal sampling for skew-resistant join size estimation. In *Proc. of the 1996 ACM SIGMOD Intl. Conf. on Management of Data*, pages 271–281, 1996.

[23] Venkatesh Ganti, Mong-Li Lee, and Raghu Ramakrishnan. ICICLES: Self-tuning samples for approximate query answering. In *Proc. of the 2000 Intl. Conf. on Very Large Data Bases*, pages 176–187, 2000.

[24] Rainer Gemulla. *Sampling Algorithms for Evolving Datasets.* PhD thesis, Technische Universität Dresden, 2008.

[25] Rainer Gemulla and Wolfgang Lehner. Sampling time-based sliding windows in bounded space. In *Proc. of the 2008 ACM SIGMOD Intl. Conf. on Management of Data*, pages 379–392, 2008.

[26] Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. A dip in the reservoir: maintaining sample synopses of evolving datasets. In *Proc. of the 2006 Intl. Conf. on Very Large Data Bases*, pages 595–606, 2006.

[27] 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.

[28] Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. Maintaining bounded-size sample synopses of evolving datasets. *The VLDB Journal*, 17(2):173–201, 2008.

[29] 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.

[30] Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *Proc. of the 2001 Annual ACM Symp. on Parallel Algorithms and Architectures*, pages 281–291, 2001.

[31] Phillip B. Gibbons, Yossi Matias, and Viswanath Poosala. Fast incremental maintenance of approximate histograms. In *Proc. of the 1997 Intl. Conf. on Very Large Data Bases*, pages 466–475, 1997.

[32] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data*, 1(1):4, 2007. ISSN 1556-4681.

[33] Peter J. Haas. Data stream sampling: Basic techniques and results. In Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi, editors, *Data Stream Management: Processing High Speed Data Streams*. Springer, 2008.

[34] Peter J. Haas and C. König. A bi-level bernoulli scheme for database sampling. In *Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of Data*, pages 275–286, 2004.

[35] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Arun N. Swami. Selectivity and cost estimation for joins based on random sampling. *Journal of Computer and System Sciences*, 52(3):550–569, 1996.

[36] M. Hadjieleftheriou, X. Yu, N. Koudas, and D. Srivastava. Selectivity estimation of set similarity selection queries. In *Proc. of the 2008 Intl. Conf. on Very Large Data Bases*, 2008. (to appear).

[37] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online aggregation. In *Proc. of the 1997 ACM SIGMOD Intl. Conf. on Management of Data*, pages 171–182, 1997.

[38] *The Expanding Digital Universe*. IDC, 2007. http://www.emc.com/digital_universe.

[39] *The Diverse and Exploding Digital Universe*. IDC, 2008. http://www.emc.com/digital_universe.

[40] Ihab F. Ilyas, Volker Markl, Peter J. Haas, Paul Brown, and Ashraf Aboulnaga. CORDS: automatic discovery of correlations and soft functional dependencies. In *Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of Data*, pages 647–658, 2004.

[41] Christopher Jermaine, Subramanian Arumugam, Abhijit Pol, and Alin Dobra. Scalable approximate query processing with the dbo engine. In *Proc. of the 2007 ACM SIGMOD Intl. Conf. on Management of Data*, pages 725–736, 2007.

[42] Ruoming Jin, Leo Glimcher, Chris Jermaine, and Gagan Agrawal. New sampling-based estimators for olap queries. In *Proc. of the 2006 Intl. Conf. on Data Engineering*, page 18, 2006.

[43] Shantanu Joshi and Christopher Jermaine. Sampling-based estimators for subset-based queries. *The VLDB Journal*, 2008. (to appear).

[44] Najmeh Joze-Hkajavi and Kenneth Salem. Two-phase clustering of large datasets. Technical Report CS-98-27, Department of Computer Science, University of Waterloo, 1998. http://www.cs.uwaterloo.ca/research/tr/1998/27/CS-98-27.pdf.

[45] Raghav Kaushik, Jeffrey F. Naughton, Raghu Ramakrishnan, and Venkatesan T. Chakravarthy. Synopses for query optimization: A space-complexity perspective. *ACM Transactions on Database Systems*, 30(4):1102–1127, 2005. ISSN 0362-5915.

[46] Per-Åke Larson, Wolfgang Lehner, Jingren Zhou, and Peter Zabback. Cardinality estimation using sample views with quality assurance. In *Proc. of the 2007 ACM SIGMOD Intl. Conf. on Management of Data*, pages 175–186, 2007.

[47] A.I. McLeod and D.R. Bellhouse. A convenient algorithm for drawing a simple random sample. *Applied Statistics*, 32(2):182–184, 1983.

[48] Frank Olken. *Random Sampling from Databases*. PhD thesis, Lawrence Berkeley Laboratory, 1993. LBL-32883.

[49] Viswanath Poosala, Peter J. Haas, Yannis E. Ioannidis, and Eugene J. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proc. of the 1996 ACM SIGMOD Intl. Conf. on Management of Data*, pages 294–305, 1996.

[50] Carl-Erik Särndal, Bengt Swensson, and Jan Wretman. *Model Assisted Survey Sampling*. Springer Series in Statistics. Springer Verlag, 1991.

[51] 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. ISSN 1041-4347.

[52] Richard Winter. Why are data warehouses growing so fast? *Beye Business Intelligence Network*, 2008. http://www.b-eye-network.com/view/7188.

[53] Fei Xu, Chris Jermaine, and Alin Dobra. Confidence bounds for sampling-based group by estimates. *ACM Transactions on Database Systems*, 2008. (to appear).