Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports

Phillip B. Gibbons

Information Sciences Research Center
Bell Laboratories
600 Mountain Avenue
Murray Hill NJ 07974
gibbons@research.bell-labs.com

Abstract

Estimating the number of distinct values is a well-studied problem, due to its frequent occurrence in queries and its importance in selecting good query plans. Previous work has shown powerful negative results on the quality of distinct-values estimates based on sampling (or other techniques that examine only part of the input data). We present an approach, called *distinct sampling*, that collects a specially tailored sample over the distinct values in the input, in a single scan of the data. In contrast to the previous negative results, our small *Distinct Samples* are guaranteed to accurately estimate the number of distinct values. The samples can be incrementally maintained up-to-date in the presence of data insertions and deletions, with minimal time and memory overheads, so that the full scan may be performed only once. Moreover, a stored Distinct Sample can be used to accurately estimate the number of distinct values within any range specified by the query, or within any other subset of the data satisfying a query predicate.

We present an extensive experimental study of distinct sampling. Using synthetic and real-world data sets, we show that distinct sampling gives distinct-values estimates to within 0%–10% relative error, whereas previous methods typically incur 50%–250% relative error. Next, we show how distinct sampling can provide fast, highly-accurate approximate answers for "report" queries in high-volume, session-based event recording environments, such as IP networks, customer service call centers, etc. For a commercial call center environment, we show that a 1% Distinct Sample provides approximate answers typically to within 0%–10% relative error, while speeding up report generation by 2–4 orders of magnitude.

**Proceedings of the 27th VLDB Conference,
Roma, Italy, 2001**

1 Introduction

Estimating the number of distinct values for some target attribute in a data set is a well-studied problem. The statistics literature refers to this as the problem of estimating the number of *species* or *classes* in a population (see [5] for a survey). The problem has been extensively studied in the database literature (e.g., [22, 23, 27, 38, 21, 29, 28, 18, 9, 7]) and elsewhere (e.g., [11, 4, 10, 19]). Estimates of the number of distinct values in a column are commonly used in query optimizers to select good query plans. In addition, histograms within the query optimizer commonly store the number of distinct values in each bucket, to improve their estimation accuracy [34, 32]. Distinct-values estimates are also useful for network monitoring devices, in order to estimate the number of distinct destination IP addresses, source-destination pairs, requested urls, etc.

Estimating the number of distinct values in a data set is a special case of the more general problem of approximate query answering of *distinct values queries*, i.e., "count distinct" queries. Approximate query answering is becoming an indispensable means for providing fast response times to decision support queries over large data warehouses. Fast, approximate answers are often provided from small synopses of the data (such as samples, histograms, wavelet decompositions, etc.) [14, 37, 3, 25, 33, 36, 1, 6, 12, 8]. Commercial data warehouses are approaching 100 terabytes, and new decision support arenas such as click stream analysis and IP traffic analysis only increase the demand for high-speed query processing over terabytes of data. Thus it is crucial to provide highly-accurate approximate answers to an increasingly rich set of queries. Distinct values queries are an important class of decision support queries, and good quality estimates for such queries may be returned to users as part of an online aggregation system [20, 17], or an approximate query answering system [14, 37, 2, 3, 25, 33, 36, 1, 6, 12, 8, 26]. Because the answers are returned to the users, the estimates must be highly-accurate (say within 10% or better with 95% confi-

```
select count(distinct target-attr)
from rel
where P
```

Figure 1: Distinct Values Query template.

```
select count(distinct o_custkey)
from orders
where o_orderdate >= '2001-01-01'
```

Figure 2: Example query matching the template.

dence), and supported by error guarantees. Unfortunately, *none* of the previous work in approximate query answering provides fast, provably good estimates for common distinct values queries.

The most well-studied approach for distinct-values estimation is to collect a uniform random sample S of the data, store S in a database, and then use S at query time to provide fast, approximate answers to distinct values queries [22, 23, 27, 21, 29, 5, 28, 18, 19, 9, 7]. However, previous work [28, 18, 9, 7] has shown powerful negative results on the quality of distinct-values estimates based on sampling (or other techniques that examine only part of the input data), even for the simple case of counting the number of distinct values in a column. The strongest negative result is due to Charikar et al. [7], who proved that estimating the number of distinct values in a column to within a small constant factor (with probability $> \frac{1}{2}$) requires that *nearly the entire data set be sampled*. Moreover, all known sampling-based estimators provide unsatisfactory results on data sets of interest [7], even for this simple case. Thus collecting a uniform sample of say 1% of the data (or otherwise reading just 1% of the data) is unable to provide good guaranteed error estimates.[1] Thus highly-accurate answers are possible only if (nearly) all the data is read when constructing the synopsis.

Distinct sampling. In this paper, we present an approach, called *distinct sampling*, that reads all the data in a single scan, and collects a specially tailored sample over the distinct values. In contrast to the above negative result, our small *Distinct Samples* can be stored in a database and used at query time to estimate the number of distinct values (and answer other common distinct values queries), with high accuracy (within 10%) and error guarantees. After the initial scan to construct it, a Distinct Sample can be incrementally maintained up-to-date as new data is inserted or deleted, with minimal time and memory overheads. Thus although we necessarily read all the data, we may do so only once.

Figure 1 depicts the common distinct values queries supported by distinct sampling. Figure 1 gives a template for distinct values queries, where `P` is an arbitrary predicate on the attributes in the relation `rel`. (We discuss joins later in Section 3.3.) A single Distinct Sample for a `target-attr` is designed to provide estimated answers to *all* queries matching the template for the given `target-attr`. We collect and maintain the Distinct Sample prior to seeing any queries; thus our Distinct Samples must be sufficiently flexible to accommodate arbitrary query predicates `P`. Figure 2 presents an example query matching this template, where the schema is the TPC-D/TPC-H/TPC-R benchmark schema. This query asks: "How many distinct customers have placed orders this year?" Unlike previous approaches, Distinct Samples can provide a highly-accurate approximate answer for this query, with error guarantees.

Using a variety of synthetic and real-world data sets, we show that distinct sampling gives estimates for distinct values queries that are within 0%–10%, whereas previous methods were typically 50%–250% off, across the spectrum of data sets and queries studied.

Event reports. We further show how distinct sampling can provide fast, highly-accurate approximate answers for "report" queries arising in high-volume, session-based event recording environments. In such environments, there is a continual stream of events to be recorded (a *data stream*). Each individual event is associated with a session: A *session* is a sequence of logically related events that occur over time. There are a large number of overlapping sessions. For example, in IP network protocols, a sequence of packets comprise a session or *flow* between a source and a destination; these packets are intermixed with the packets for other sessions. In the telecommunications arena, a sequence of call-processing events comprise the session associated with connecting and completing a phone call. In the customer service arena, a sequence of call-handling events (a customer dials in, is put on hold, an agent answers, etc.) comprise the handling of a single customer's call. The event recording environment creates a log record for each event as it occurs; the record is timestamped and tagged with a unique id for its session.

To keep up with the rapid rate of incoming events, the event records are simply dumped to append-only databases. As a continuously running background process, the logs from these databases are sent to a central data warehouse, for data analysis. Precanned reports are executed by the hour or even by the minute, tracking the progress of events and sessions, system utilization, possible anomalies, etc. (Ad hoc queries are less frequent.) The importance of fast processing of precanned report queries is evidenced by the recent TPC-R benchmark standard for report queries. Due to the high volume of data and stringent response time requirements, it is very desirable to have fast, highly-accurate *approximate* reports.[2] Unfortunately, *none* of the previous work in approximate query answering provides fast, highly-accurate approximate reports for session-based event recording environments.

We show how distinct sampling can be used to provide fast, highly-accurate approximate reports. As we shall see, Distinct Samples are effective in this domain because – unlike uniform random samples – if *some* event from a session is in the sample, then *all* events from that session are also in the sample. (This is accomplished despite the fact

[1] Approaches based on histograms or wavelets [37, 25, 33, 36, 12] likewise fail to provide good error guarantees for distinct values queries.

[2] In fact, often only a few digits of precision are reported even for the *exact* report, so an approximate answer that is accurate to this precision is *indistinguishable* from an exact answer.

that events for multiple sessions are intermixed.) For a real-world customer service (*call center*) event recording environment, we show that a 1% Distinct Sample gives estimates typically to within 0%–10% relative error, while speeding up report generation by 2–4 orders of magnitude.

This work was done in the context of the Aqua approximate query answering system [2, 3, 1].

Outline. The remainder of this paper is organized as follows. Section 2 discusses related work in further detail. In Section 3, we describe our distinct sampling approach. Section 4 highlights our extensive experimental study. Due to page constraints, most of the experimental results appear only in the full paper [13].

2 Related Work

Sampling-based approaches. As indicated above, there is an extensive literature on distinct-values estimation from samples [22, 23, 27, 21, 29, 5, 28, 18, 19, 9, 7]. The large number of different approaches studied reflects the difficulty of the problem; this difficulty was first proved formally by Chaudhuri et al. [9], and later extended by Charikar et al. [7], who proved the following theorem:

Theorem 1 [7] *Consider any (possibly adaptive and randomized) estimator $\hat{D}$ for the number of distinct values D that examines at most r rows in a table with n rows. Then, for any $\gamma > e^{-r}$, there exists a choice of the input data such that with probability at least γ, error$(\hat{D}) \geq \sqrt{\frac{n-r}{2r} \ln \frac{1}{\gamma}}$.*

The error metric is the *ratio error*: *error*$(\hat{D}) = \max\left(\frac{D}{\hat{D}}, \frac{\hat{D}}{D}\right)$. They also present an extensive experimental comparison of a number of the best approaches, including two estimators they propose – GEE and AE – which are discussed further in Section 4.

This extended research focus on sampling-based estimators is due in part to two factors. First, in the applied statistics literature, the option of collecting data on more than a small sample of the population is generally not considered, because there is typically a prohibitive expense for doing so (e.g., for collecting medical data on every person in the world, or every animal of a certain species). Second, in the database literature, where scanning the entire data set is viable, it is generally considered too expensive. Echoing other works, Charikar et al. [7] argue that extracting a sample, and then estimating from that sample, is the *only scalable approach* to distinct-values estimation. In contrast, we would argue that one pass algorithms with fast incremental maintenance (that accommodate updates without revisiting the existing data set) also provide a scalable approach, whenever such algorithms exist.

One pass approximate counting. Flajolet and Martin [11] pioneered one-pass approximate distinct counting, with an algorithm that hashes sets of data values to bit vectors, for counting purposes. Each distinct value in the domain gets mapped to bit i in the vector with probability $2^{-(i+1)}$. To obtain a good estimator, Flajolet and Martin take the average over tens of applications of this procedure (with different hash functions). Alon et al. [4] also considered a similar scheme. Palmer et al. [30, 31] showed how to extend this one pass counting to speed up by 2–3 orders of magnitude the computation of the *Neighbourhood* function for massive graphs, e.g., to analyze the connectivity and fault tolerance of the Internet. Whang et al. [38] proposed and studied a technique called *linear counting*, which also runs in one pass and produces an estimate with any user specified accuracy, using hashing. They show that a load factor (number of distinct values/hash table size) of 12 can be used for estimating to within 1% error, using a hash function that maps each distinct value in the domain uniformly at random to a number in $[1, m]$. Because the number of distinct values can be as large as the data size N, the space m must be large as well ($\Theta(N)$), so this approach fails to provide a small space synopsis. Cohen [10] proposed and studied a size estimation framework, useful for estimating the number of descendants of each node in a directed graph, and related problems. At its core, this framework is performing distinct counting, to ensure that descendants reachable by multiple paths are not double counted. In the approach, sets of data values are hashed to the minimum element of a random permutation of the values, for counting purposes. This approach requires tens of repeated applications, and does not handle deletions of data items.

Each of the one pass approaches above provides an estimator for a single target attribute. However, unlike distinct sampling, their respective hashing schemes destroy any hope of being able to estimate the number of distinct values over a subset of the data specified by a subsequent predicate. Thus they fail to provide estimators for the more general distinct values queries of Figure 1. Moreover, they are no help in producing approximate reports.

Gibbons and Tirthapura [16] considered a distributed setting in which each party observes a set of items, and the goal is to estimate the size of the union of all the sets. The parties have limited memory and can only communicate with one another after all parties have observed their respective sets. This work did not focus on distinct values queries or database scenarios, and provided no experimental evaluation. Distinct sampling builds upon this previous work, extending their algorithm to handle distinct values queries with predicates, as well as approximate reports. Moreover, we present techniques for incremental maintenance, and provide an extensive experimental comparison with previous sampling-based approaches.

3 Distinct Sampling

Our goal is to collect and store information on database tables that will permit highly-accurate estimates for distinct values queries on these tables. For simplicity, we will describe our approach focusing on a single target attribute (`target-attr`) from a single table (`rel`), as in Figure 1. The approach extends easily to multiple target attributes, and we will later briefly discuss extensions to handle joins. We assume there is an *a priori* bound on the storage set aside for our purposes, e.g., space for a sample of 10K rows

from a large table.

3.1 Problems To Overcome

If we sample uniformly from the rows in `rel`, as a traditional sample would, we will likely miss attribute values that occur only rarely in `rel`. Instead, we would like to ensure that *all* distinct values are represented in our sample. For attributes with many distinct values, however, there may not be room to store all the distinct values within the given storage bound. In such cases we would like to collect a uniform random sample of the distinct values. This has advantages detailed below. Moreover, we need to know how to scale the answers produced from this uniform sample, so we need to store the sampling rate.

Two problems arise. First, we need to detect the first occurrence of a value as the table is scanned. For each row, we can certainly check the current sample to see if the row has a value for `target-attr` that is already in the sample. But we have insufficient storage to keep track of all distinct values, so what should be done if the value is not in the current sample? We have no idea whether this is the first time we have seen this value or we have seen it before but ignored it (and hence for consistency we must ignore it again). Adding the value to the sample with some probability would *not* produce a uniform sample of the distinct values, and in fact would be quite biased towards frequently occurring values [14]. On the other hand, ignoring the value is no good either, because this would completely bias the sample towards values occurring early in the table. Second, even if we could detect the first occurrence of a value, how would we know what sampling rate to use to decide whether or not the value should be included in the sample? The rate depends on the total number of distinct values, which is the problem we are trying to solve in the first place.

Moreover, our goal is not only to produce an estimate of the total number of distinct values, but also estimate distinct values queries over subsets of the data selected by subsequent query predicates. Thus simply maintaining all (or a sample of) the distinct values themselves is insufficient, because it does not permit proper accounting for subsequent predicates (which may often be on other attributes).

3.2 Our Solution: Distinct Sampling

We now describe *distinct sampling*, and show how it overcomes these problems. In a Distinct Sample:

- Each distinct `target-attr` value in the table `rel` is equally likely to be in the sample.
- For each distinct value v in the sample, we have:
  - a count of the number of times v appears in the *entire table* scanned thus far, and
  - either *all* the rows in the table with `target-attr` $= v$, or, if there are more than t such rows, a *uniform sample* of t of these rows (t is a parameter).

Distinct Sampling(space bound B, valSampSize t)
1. initialize $\ell := 0$, $S := \emptyset$
2. select the hash function die-hash
3. do while (more rows R to scan) {
4. die-level := die-hash(v), for target-attr value v in R
5. if (die-level $\geq \ell$) {
6. if (v appears 0 to $t-1$ times in S) {
7. add row R to S
8. if (v now appears t times)
9. add to S a dummy row for v with $c_v := t$
}
10. else { // v appears exactly t times
11. retrieve the count c_v from the dummy row for v
12. increment c_v
13. with probability t/c_v, add row R to S
and evict a random row with value v from S
}
14. if ($|S| \geq B$) { // S is full
15. evict from S all rows with target-attr values w
such that die-hash(w) $= \ell$
16. increment ℓ
}
}
}

Figure 3: The Distinct Sampling Algorithm

The distinct sampling algorithm is given in Figure 3. The algorithm has two parameters: the bound B on the available sample size (the number of rows in total) and the bound t on the maximum number of rows to retain for a single distinct value. The basic idea is as follows. (An example is given below.) There is a *level* associated with the procedure, that is initially 0 but is incremented each time the sample bound B is reached. Each value in the domain is mapped to a random level, called its *die-level*. An easily computed hash function is used, so that each time a given value occurs in the table, it maps to the same die-level (Step 4). We scan through the table, only retaining rows in our Distinct Sample S whose target-attr value's die-level is at least as large as the current level ℓ (Step 5). Eventually either we reach the end of the table or we exhaust all the space available for S. In this latter case, we create more room in S by evicting all rows with die-level ℓ and then incrementing ℓ (Steps 14-16). We then proceed with the scan.

The invariant we maintain throughout is that S contains *all* the distinct values appearing in the scanned portion of the table whose die-level is at least ℓ, and no other distinct values. Because levels for distinct values are chosen at random, S contains a *uniform sample* of the distinct values in the scanned portion. (Of course, it may have many rows duplicating the same values.)

To expedite the sampling and subsampling, our hash function (called *die-hash*) maps the value domain onto a logarithmic range, such that each distinct value gets mapped to i with probability $2^{-(i+1)}$ (i.e., mapped to 0 with probability $\frac{1}{2}$, to 1 with probability $\frac{1}{4}$, etc.). Thus a current level of ℓ indicates that only a $2^{-\ell}$ fraction of the domain is currently eligible for S. It follows that if S is the

Rows scanned			Result	
row	attr value	die level	distinct sample	level ℓ
R_1	5	0	$\{R_1\}$	0
R_2	3	2	$\{R_1, R_2\}$	0
R_3	3	2	$\{R_1, R_2, R_3\}$	0
R_4	8	0	$\{R_1, R_2, R_3, R_4\}$	0
R_5	2	1	$\{R_1, R_2, R_3, R_4, R_5\}$	0
R_6	7	0	$\{R_1, R_2, R_3, R_4, R_5, R_6\}$	0
R_7	8	0	$\{R_2, R_3, R_5\}$	1
R_8	3	2	$\{R_2, R_3, R_5, R_8, c_3 = 3\}$	1
R_9	3	2	$\{R_2, R_5, R_8, R_9, c_3 = 4\}$	1
R_{10}	5	0	$\{R_2, R_5, R_8, R_9, c_3 = 4\}$	1
R_{11}	3	2	$\{R_2, R_5, R_8, R_{11}, c_3 = 5\}$	1
R_{12}	9	1	$\{R_2, R_5, R_8, R_{11}, R_{12}, c_3 = 5\}$	1

Figure 4: An example run of the Distinct Sampling Algorithm, for $B = 7$ and $t = 3$.

Distinct Sample and ℓ is the current level after the entire table has been scanned, then the number of distinct values in the table can be estimated as 2^ℓ times the number of distinct values in S.

In our algorithm description thus far, we have implied that we retain in S all rows whose target-attr value's die-level is at least ℓ. However, doing so could swamp S with many rows having the same distinct value, leaving little room for other distinct values. Instead, we place a limit, valSampSize $= t$, on the number of rows with the same value. For values that reach that limit, we use reservoir sampling [35] to maintain a uniform sample of the t rows with that value (Steps 10-13). Reservoir sampling requires knowing the number of rows with the same value thus far (whether currently in the Distinct Sample or not); thus we also store a dummy row in the Distinct Sample that contains the exact number of occurrences of this value thus far (Step 9).

An example algorithm execution is depicted in Figure 4, for a 12 row table. There are three distinct values in the resulting Distinct Sample (2, 3, and 9), so the number of distinct values in this table is estimated as $3 \cdot 2^1 = 6$, which matches the actual number of distinct values.

3.3 Discussion and Further Details

Setting the parameter t. For queries without predicates, or queries with predicates only on the target attribute, the valSampSize t could be set to 1, resulting in the best performance for distinct sampling on such queries, because the available storage would be devoted entirely to distinct values (and not to multiple rows with the same value). However, our goal is to estimate distinct values queries for more general predicates. A good rule of thumb is to set t to be the minimum of (i) twice the inverse of the minimum predicate selectivity, q, for queries of interest (e.g., 100 for predicates down to 2%) and (ii) 2% of the sample size B:

$$t = \min(2/q, B/50) \quad (1)$$

Because we store *all* rows for a value, up to t, we know precisely whether any of the rows with this value satisfy the predicate. When the number of rows with this value exceeds t, we have a uniform sample of these rows, of size t; thus an expected $q \cdot t$ of these rows will satisfy the predicate, where q is the selectivity of the predicate over all the table rows with this value. When $t \geq \frac{2}{q}$, we expect at least two rows in the sample to satisfy the predicate. On the other hand, if no rows satisfy the predicate, then no rows in the sample satisfy the predicate, as desired for accurate estimation.

The hash function die-hash. We now discuss the details of the hash function die-hash. For simplicity, assume the `target-attr` domain is the set of integers in $[0..D-1]$, where D is a power of two. Let $m = \log_2 D$. For every value v, die-hash(v) is a mapping from $[0..D-1]$ to $[0..m]$, such that, independently for each v,

$$\forall \ell \in [0..m-1] : \mathbf{Pr}\{\text{die-hash}(v) = \ell\} = 2^{-(\ell+1)} \quad (2)$$

Three parameters define a particular die-hash: α, β, and hashmod. We set hashmod to be D. We choose α uniformly at random from [1..hashmod-1] and β uniformly at random from [0..hashmod-1]. For any x in [0..hashmod-1], define LeadZeros(x) to be the number of leading zeros in x when viewed as a $\log_2$(hashmod)-bit number. Then for each value v encountered during the scan, die-hash is computed as:

$$\text{die-hash}(v) = \text{LeadZeros}((\alpha \cdot v + \beta) \bmod \text{hashmod})$$

This hash function was used in [4], where it was shown to satisfy equation 2 and pairwise independence among the values v.

Storing by die-level. We can store the Distinct Sample using an array L of size $\log D$, such that $L[i]$ is the head of a linked list of all the rows in the Distinct Sample with die-level i. This makes finding the rows to evict during a level change trivial. In fact, we can use lazy deletion, reclaiming the sample slots associated with the most recently evicted level one-at-a-time, as new slots are needed. This has the advantage that the processing time for each row is more consistent: there are no longer slow steps due to level changes.

Producing an estimate. Given a stored Distinct Sample S and its current level ℓ, we execute the distinct values query on S, remembering to ignore the dummy rows, and multiply the result by the scale factor 2^ℓ.

Accuracy guarantees. The following theorem presents the accuracy guarantees for distinct sampling. The guarantees are dependent on the selectivity of the predicates in the queries of interest, in a somewhat non-trivial way. Specifically, let V be the set of distinct `target-attr` values in the relation, and, for any predicate P, let $V_P \subseteq V$ be the set of distinct values in rows satisfying P. We define the *target selectivity*, $q_{tar} \leq 1$, to be $|V_P|/|V|$, i.e., the number of distinct values satisfying P divided by the total number of distinct values in the relation. Next, we consider only values v in V_P, and define the *value selectivity for* v, $q_{val}(v) \leq 1$, to be the number of rows with `target-attr` value v satisfying P divided by the total number of rows

with value v in the relation. (Note that $q_{val}(v) > 0$ because $v \in V_P$.) Then let the overall *value selectivity*, $q_{val} \leq 1$, be the median of the $q_{val}(v)$.[3] Finally, let Q be the set of all distinct values queries matching the template in Figure 1 for a given `target-attr`, with target selectivity at least q^*_{tar} and value selectivity at least q^*_{val}.

Theorem 2 *For any positive $\epsilon \leq 1$ and $\delta \leq 1$, a single Distinct Sample for the* `target-attr`*, with $t = \Theta\left(\frac{\log(1/\delta)}{q^*_{\text{val}}}\right)$ and $B = \Theta\left(\frac{t \cdot \log(1/\delta)}{q^*_{\text{tar}} \cdot \epsilon^2}\right)$, provides an estimate for* any *query in Q such that the estimate is guaranteed to be within a relative error ϵ with probability $1 - \delta$.*

The proof appears in the full technical report [13], and assumes fully independent random hash functions.

Handling joins. Distinct sampling over joins suffers from the same difficulties as uniform sampling over joins [3], and the same *join synopsis* approach can be used to handle the common case of multiway foreign key joins. However, the following simpler approach can work as well: Often, fast approximate answers can be obtained by replacing the large fact table in the query with a small synopsis of that table, leaving all other relations in the query unchanged. Even in the presence of joins, the resulting query runs much faster, because the size of the fact table is the performance bottleneck in the original query. Under such scenarios, a Distinct Sample on a `target-attr` in the large fact table can provide the same accuracy for queries with joins as without joins.

Incremental maintenance. The Distinct Sampling algorithm in Figure 3 is well-suited to incremental maintenance. Besides the Distinct Sample itself, we need to save only the current level ℓ, the value of t, and the two coefficients and the modulus which define die-hash. Each insertion is handled by executing one loop (Steps 4-16) of the algorithm. Note that for all but a small fraction of the insertions (namely, $\frac{1}{2^t}$ on average), the test in Step 5 fails and no further processing is required. Similarly, we can ignore all but a small fraction of the updates and deletions (after we compute their die-levels). Each deletion with a die-level at least ℓ must have a matching row in S (which can be safely deleted), or its attribute value is part of a reservoir sample for that value, in which case we apply the technique in [15] for deleting from reservoir samples. (Note that if a substantial fraction of the relation is deleted, it may be necessary to rescan the relation in order to preserve the accuracy guarantees [13].) Each update with a die-level at least ℓ, which does not modify the target attribute, is handled by updating the row in place, if it is present in S. Updates that change the target attribute are handled as a deletion followed by an insertion. To minimize overheads, insertions, deletions, and updates can be processed in batches.

Trade-offs with traditional sampling. There are several advantages to traditional samples over Distinct Samples. One is that a traditional sample is much faster to construct: it can be extracted without scanning the entire data set. Moreover, a *single* sample can be used for distinct-values estimation for *any* target attribute (although the accuracy is often poor), whereas distinct sampling requires a separate Distinct Sample for each target attribute (and a priori knowledge of the target attributes). All such Distinct Samples can be collected in one scan of the data, but the update costs increase with the number of target attributes. Because any combination of attributes may comprise a (composite) target attribute for a distinct values query, there can be a large number of such target attributes. On the other hand, our experimental results show that we can often store Distinct Samples for tens of attributes within the same space as a single traditional sample, and still obtain far more accurate results. Moreover, the speed of answering a query from a stored sample depends on the size of the sample used, not the overall size of all samples collected. Thus having multiple Distinct Samples does not slow down query response-time. Hence, in practice, one may wish to have Distinct Samples for a dozen or so of the most important target attributes (single attributes or attribute combinations), and then resort to a traditional sample for any other target attributes.

[3] More generally, any q_{val} no larger than a constant fraction of the $q_{val}(v)$ suffices.

4 Experimental Evaluation

In this section, we highlight the results of an experimental evaluation of distinct sampling. (Our complete set of results is in the full paper [13].) Using synthetic and real-world data sets, we show that distinct sampling gives estimates to within 0%–10% whereas previous methods were typically 50%-250% off.

4.1 Algorithms studied

We compare estimates obtained using a Distinct Sample to estimates obtained from a uniform random sample by applying two of the new estimators, GEE and AE, proposed in [9, 7]. These estimators, especially AE, perform as well or better than the other estimators, across a range of synthetic and real-world data sets [7]. Descriptions of GEE and AE, including straightforward extensions to handle predicates, can be found in the full paper [13]. Note that in all cases, the query predicate P is not known at the time the Distinct Sample or uniform sample is collected.

In our experiments, each algorithm is given the same sample size. Most of our experiments compare the three estimators for a range of sample sizes. For distinct sampling, we set valSampSize $= 100$ for all our experiments with Zipf distributions and valSampSize $= 50$ for all our experiments with real-world data. (These settings are based on equation 1: We use a smaller setting for the real-world data because the data sizes and hence the sample sizes are smaller.)

The experiments were run on a 733 MHz Pentium PC with 256MB RAM running Linux. Each data point plotted is the average of 7 independent trials.

To facilitate comparison with [7], we evaluate an estimate using the error metric they study — the *ratio er-*

ror metric: If A is the exact answer for the distinct values query, and $\hat{A}$ is the estimate, then the ratio error is $error(\hat{A}) = \max(\frac{A}{\hat{A}}, \frac{\hat{A}}{A})$. Note that the ratio is always ≥ 1, and the closer to 1 the better. In the text, we often find it revealing to convert a ratio error r into a *percentage error*, which we define to be $(r-1) \cdot 100\%$. For overestimates, this is exactly $\frac{|\hat{A}-A|}{A} \cdot 100\%$, the standard definition of percentage relative error.

4.2 Experimental Results: Synthetic Data

We first studied data sets generated from Zipfian distributions. These enable us to compare the accuracy of the various methods in a controlled manner. We generated values from Zipf(z) for zipf parameter z ranging from 0 (the uniform distribution) to 4 (very high skew). Note that this also varies the number of distinct values from high (when $z = 0$) to low (when $z = 4$).

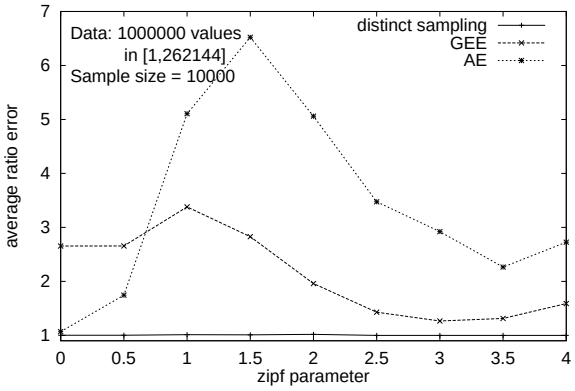

Figure 5: Accuracy vs. data skew

Accuracy vs. skew. Our first experiment (Figure 5) studies the accuracy of the three estimators over a range of skew in the data set. We generate 1 million values from a Zipf(z) distribution, for $z = 0.0, 0.5, 1.0, 1.5, \ldots, 4.0$. The sample size for both the Distinct Sample and the traditional sample used by GEE and AE is fixed at 10K, which is a 1% sample. The distinct sampling estimator has an average ratio error of less than 1.02 (i.e., less than a 2% relative error) across the entire range of skew. An examination of the data values plotted reveals that the average ratio error for distinct sampling is 1.0024 (0.2% error) for uniform data, increasing to 1.019 for $z = 2$, and then back down to exactly 1 for high skew ($z \geq 2.5$). Distinct sampling produced an exact answer for the $z \geq 2.5$ data sets because there was sufficient space to fit a sample of size valSampSize for all the distinct values in the data set. The GEE estimator was much worse, ranging between 1.26 (i.e., 26% error) and 3.37 (i.e., off by more than a *factor* of 3). Such large errors are likely unacceptable for approximate answers returned in response to user queries. The AE estimator was better than the GEE at low skew, with ratios of 1.06 (6% error) and 1.74 (74% error) for the uniform and $z = 0.5$ distributions, but then much worse at moderate to high skew (up to a 6.5 ratio). In summary, the percentage errors for both GEE and AE are at least 25 times larger than for distinct sampling at every single skew value.

data set	number of rows	target attribute	domain size	num. of distinct
wuthering	120,951	words	128K	10,545
covtype	581,012	elevation	8K	1,978
census	48,842	native-country	64K	42
call-center	418,921	workitem-id	1M	46,691

Table 1: Data set characteristics

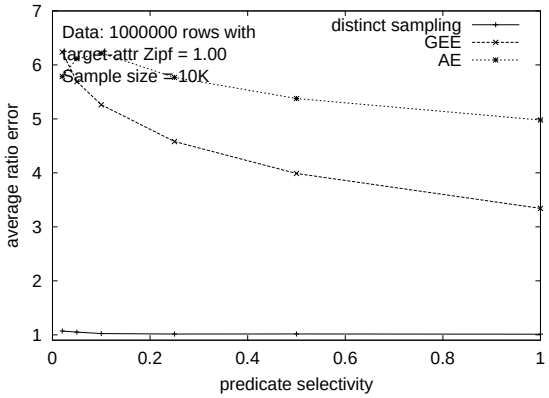

Figure 6: Accuracy vs. predicate selectivity

Accuracy vs. predicate selectivity. Our next experiment (Figure 6) shows that distinct sampling again achieves very low ratios (1.01 to 1.07) compared with both GEE (3.34 to 6.23) and AE (4.98 to 6.22), across a range of predicate selectivities, from 2% to 100%. The predicate was a one-sided range predicate on a non-target attribute, and the sample was a 1% sample.

Other zipf experiments. In the full paper, we report on experiments varying the data size from 100K to 1 million, varying the sample sizes from 0.2% to 6.4%, and varying the number of simultaneous Distinct Samples sharing the target space bound from 1 to 32. The results were qualitatively the same as those discussed above.

4.3 Experimental Results: Real-World Data

We next studied three real-world data sets, summarized in Table 1. (The fourth, call-center, is studied in Section 4.4.) The results were similar to the zipfian data sets, with the distinct sampling estimator doing quite well, while the other two estimators having large errors.

Wuthering Heights. *Wuthering* is a data set of the words in Wuthering Heights, in the order they appear in the text. As can be seen from Table 1, there are 120K words of which 10K are distinct. Figure 7 compares the three estimators, for sample sizes ranging from 500 (0.4%) to 8000 (16.5%). The distinct sampling estimates ranged from 1.08 down to 1.017, which were far better than both the GEE estimates (ranging from 2.86 down to 1.59) and the AE estimates (ranging from 8.11 down to 1.66). Note that, as in the experiments on synthetic data, the distinct sampling error at the smallest sample size is much better (by a factor

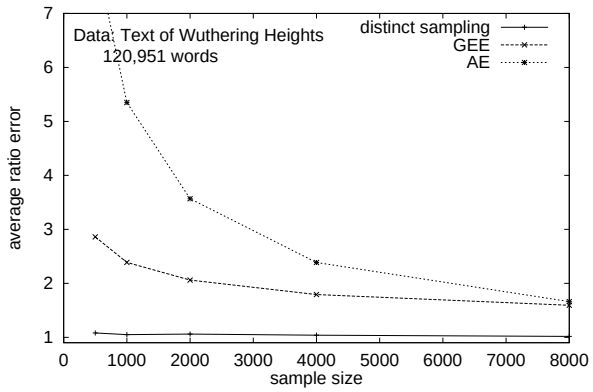

Figure 7: Accuracy vs. sample size for the text of Wuthering Heights

```
select count(distinct elevation)
from covtype.data
where elevation > 2500 and cover-type = SpruceFir
```

Figure 8: Cover Type query

of 7) than the GEE or AE errors at the largest sample size, despite the factor of 40 range in sample sizes. Thus even if the sample space B were to be divided evenly among 40 Distinct Samples targeting 40 different target attributes, Distinct sampling has a factor of 7 smaller error.

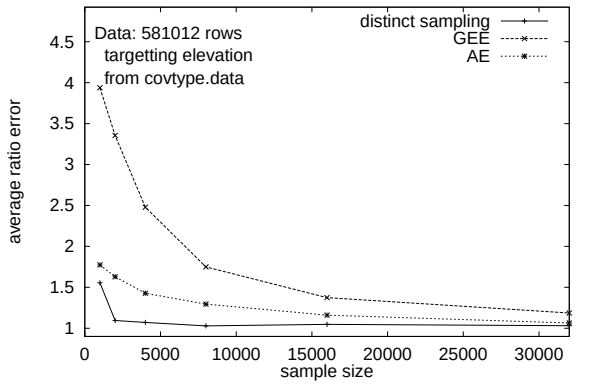

Figure 9: Accuracy vs. sample size for the Cover Type query

Cover Type. *Covtype* is the Forest Covertype data set from the National Forest Service, down-loaded from U.C. Irvine [24]. Each tuple has 54 attributes (elevation, slope, distance to highway, forest cover type, etc). Many of the attributes are low cardinality; we chose elevation, with nearly 2K distinct values, in order to study an attribute with moderate cardinality. We considered the query in Figure 8: "At how many distinct elevations above 2500 meters was there a SpruceFir forest?". We consider sample sizes of $1K, 2K, 4K, 8K, 16K, 32K$, which ranges from a 0.17% sample to a 5.5% sample. Note that the predicate restricts not just the target-attr (i.e., elevation), but also another attribute (i.e., cover-type). The selectivity of the Cover Type

```
select count(distinct native-country)
from census.data
where workclass = 'Federal-gov' or
workclass = 'State-gov' or workclass = 'Local-gov'
```

Figure 10: Census query

query predicate is 36.5%. In terms of the distinct elevations, 1161 out of 1978 satisfy the predicate (= 58.7%). Figure 9 presents the results of this experiment. At the smallest sample size, the distinct sampling estimator has a ratio of 1.56. This reflects the fact that the predicate reduces the effective sample size by almost a factor of 3. At the second smallest size (0.34% sample), the ratio improves to 1.09, and continues to improve down to 1.03 with increasing sample sizes. The GEE ratios range from 3.93 down to 1.18. The AE ratios range from 1.77 down to 1.06. Thus the distinct sampling errors are consistently smaller than the GEE and AE errors, but not by as large of margin as in previous experiments.

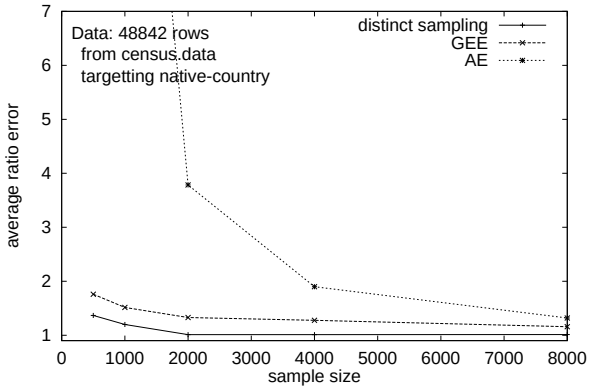

Figure 11: Accuracy vs. sample size for the Census query

Census. *Census* is a data set from the U.S. census database, also down-loaded from U.C. Irvine [24]. Each tuple has 15 attributes (age, workclass, marital status, etc). Most of the attributes have only tens of distinct values; we chose native-country (with 42 distinct values) as a representative attribute. We will consider the query in Figure 10: "How many distinct nationalities are represented by people working in government?" The query tests the effectiveness of the estimators when there are only a small number of distinct values. We consider sample sizes of $500, 1000, 2000, 4000, 8000$, which ranges from a 0.1% sample to a 1.6% sample. The selectivity of the Census query predicate is 13.4%. In terms of the distinct native-country values, 39 out of 42 satisfy the predicate (= 85.7%). At the smallest sample size, the distinct sampling ratio is 1.36, due to the low predicate selectivity. This drops to 1.2 for the 0.2% sample, and then below 1.013 (within 1.3% error) for all larger sample sizes. The GEE estimator ranges from 1.76 down to 1.15, and is consistently better than the AE estimator. The AE estimator drops dramatically from 118 to 16 (both off the plot) to 3.7 to 1.8 to 1.3. For each sample size 0.8% or larger, the AE percentage errors are

over 12 times larger than the distinct sampling percentage errors, but the difference is less for the smaller sample sizes.

To summarize, distinct sampling performs quite well for these real-world data sets, even in the presence of predicates, and far better than GEE and AE, which tend to alternate as to which is better.

4.4 Experimental Results: Call Center Reports

Call-center is a privately-available data set, obtained with permission. It is a collection of event records tracking the workflow of individual calls into a customer service center ("call center"). Records associated with an individual call are all tagged with the same workitem-id, unique to that call, and are intermixed with all other records. This data set is 10 hours of simulated call activity used in an actual test run of an alpha-version of a commercial call center processing system. This studies the high cardinality case (46K distinct workitem-ids). Note that call centers for large companies can generate $\frac{1}{2}$ gigabyte or more of event data per week, causing the report generation to be slow for the more complicated reports.

In the full paper [13], we report on a series of experiments with two real-world reports on the call-center data set. These experiments were run on the same 733 MHz Pentium PC with 256MB RAM as above, but this time running Windows NT 4.0. The call-center data was loaded into Microsoft SQL Server 7.0. Traditional samples and Distinct Samples of various sizes (1% to 25%) were extracted from the central event table, and stored in the same SQL Server database as the original data. It took only a few minutes to extract and store *all* the samples used in our experiments. The reports were developed for execution from within Seagate Crystal Reports, a commercial report generation application. The reports were rewritten to query the samples instead of the full event table.

The two reports studied are typical for session-based event reports in that *all* the events for a call session must be in the sample in order to compute the derived statistics for that call. Accordingly, for our Distinct Samples, we set the valSampSize to be larger than the maximum number of events for a call ($t = 25$). In contrast, traditional samples suffered dramatically by having only some of the events for any given session, as discussed in the full paper.

Table 2 depicts the query times and relative errors in the answers for each field in the *Call Center Performance Report* (one of the reports studied), for Aqua reports using a range of Distinct Sample sizes (a single trial each). Highly-accurate answers are obtained in orders of magnitude less time than producing the exact report.

The second report studied, the *Call Center Time of Day Performance Report*, suffered dramatically from substantial memory thrashing when producing an exact report. On our experimental platform, the exact report took 5 hours and 26 minutes to generate, whereas an Aqua report using a 5% Distinct Sample took only 10 seconds to generate, while producing answers within a 7% relative error on average.

As detailed in the full paper, the results provide a compelling case for the effectiveness of distinct sampling for speeding-up call center reports.

5 Conclusions

Distinct values queries are commonplace both within the query optimizer and in user queries and reports. As multi-terabyte data recording and warehousing environments become increasingly the norm, complex queries over such environments suffer from increasingly slow response times. Thus obtaining fast approximate answers becomes an important, attractive option for users. Previously, there were no effective techniques for providing fast approximate answers to the broad range of distinct values queries considered in this paper. This paper has presented a novel approach, called distinct sampling, for providing fast, highly-accurate approximate answers to distinct values queries for a given target attribute. Our experiments show that distinct sampling provides estimates to within 0%–10% relative error, whereas previous methods were typically 50%–250% off, for the same storage space. Moreover, Distinct Samples can be computed in one pass over the data, and incrementally maintained, with low overheads. Finally, we demonstrated how distinct sampling can provide approximate event reports that are also within 0%–10% relative error, while speeding up report generation by 2–4 orders of magnitude. Therefore, distinct sampling is the first effective technique for providing fast, approximate answers to distinct values queries and session-based event reports, and is recommended for large data recording and warehousing environments.

References

[1] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 487–498, May 2000.

[2] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua approximate query answering system. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 574–576, June 1999. Demo paper.

[3] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 275–286, June 1999.

[4] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. 28th ACM Symp. on the Theory of Computing*, pages 20–29, May 1996.

[5] J. Bunge and M. Fitzpatrick. Estimating the number of species: A review. *J. of the American Statistical Association*, 88:364–373, 1993.

[6] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *Proc. 26th International Conf. on Very Large Data Bases*, pages 111–122, September 2000.

[7] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards estimation error guarantees for distinct values. In *Proc. 19th ACM Symp. on Principles of Database Systems*, pages 268–279, May 2000.

Report	Query time	WO	WH	ABD	ABD Rate	Avg PPT	Avg SA	Avg HT
Exact	216 secs	184,753	184,753	0	0.00%	00:34	03:03	00:32
Aqua 25%	51 secs	0.2%	0.2%	0.0%	0.0%	0.0%	0.0%	0.0%
Aqua 20%	43 secs	0.5%	0.5%	0.0%	0.0%	0.0%	5.5%	0.0%
Aqua 10%	21 secs	0.4%	0.4%	0.0%	0.0%	0.0%	4.4%	0.0%
Aqua 5%	10 secs	0.004%	0.004%	0.0%	0.0%	0.0%	3.3%	0.0%
Aqua 1%	2 secs	2.6%	2.6%	0.0%	0.0%	0.0%	32.8%	0.0%

Table 2: Query times and answer accuracy vs. Distinct Sample sizes.

[8] S. Chaudhuri, G. Das, and V. Narasayya. A robust, optimization-based approach for approximate answering of aggregate queries. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 295–306, May 2001.

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

[10] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. of Computer and System Sciences*, 55(3):441–453, 1997.

[11] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Computer and System Sciences*, 31:182–209, 1985.

[12] V. Ganti, M.-L. Lee, and R. Ramakrishnan. ICICLES: self-tuning samples for approximate query answering. In *Proc. 26th International Conf. on Very Large Data Bases*, pages 176–187, September 2000.

[13] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. Technical report, Bell Laboratories, Murray Hill, New Jersey, June 2001.

[14] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 331–342, June 1998.

[15] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proc. 23rd International Conf. on Very Large Data Bases*, pages 466–475, August 1997.

[16] P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proc. 13th ACM Symp. on Parallel Algorithms and Architectures*, July 2001.

[17] P. Haas and J. Hellerstein. Ripple joins for online aggregation. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 287–298, June 1999.

[18] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proc. 21st International Conf. on Very Large Data Bases*, pages 311–322, September 1995.

[19] P. J. Haas and L. Stokes. Estimating the number of classes in a finite population. *J. of the American Statistical Association*, 93:1475–1487, 1998.

[20] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 171–182, May 1997.

[21] W.-C. Hou, G. Özsoyoğlu, and E. Dogdu. Error-constrained COUNT query evaluation in relational databases. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 278–287, May 1991.

[22] W.-C. Hou, G. Özsoyoğlu, and B. K. Taneja. Statistical estimators for relational algebra expressions. In *Proc. 7th ACM Symp. on Principles of Database Systems*, pages 276–287, March 1988.

[23] W.-C. Hou, G. Özsoyoğlu, and B. K. Taneja. Processing aggregate relational queries with hard time constraints. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 68–77, June 1989.

[24] Information and Computer Science, University of California at Irvine. ftp://ftp.ics.uci.edu/pub/machine-learning-databases, 2000.

[25] Y. Ioannidis and V. Poosala. Histogram-based techniques for approximating set-valued query-answers. In *Proc. 25th International Conf. on Very Large Databases*, pages 174–185, September 1999.

[26] I. Lazaridis and S. Mehrotra. Progressive approximate aggregate querie with a multi-resolution tree structure. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 401–412, May 2001.

[27] J. F. Naughton and S. Seshadri. On estimating the size of projections. In *Proc. 3rd International Conf. on Database Theory*, pages 499–513, 1990.

[28] F. Olken. *Random Sampling from Databases*. PhD thesis, Computer Science, U.C. Berkeley, April 1993.

[29] G. Ozsoyoglu, K. Du, A. Tjahjana, W. Hou, and D. Y. Rowland. On estimating COUNT, SUM, and AVERAGE relational algebra queries. In *Proc. Conf. on Database and Expert Systems Applications*, pages 406–412, 1991.

[30] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. Fast approximation of the "neighbourhood" function for massive graphs. Technical Report CMU CS-01-122, Carnegie-Mellon University, Pittsburgh, PA, 2001.

[31] C. R. Palmer, G. Siganos, M. Faloutsos, C. Faloutsos, and P. B. Gibbons. The connectivity and fault-tolerance of the internet topology. In *Proc. 2001 Workshop on Network-Related Data Management (NRDM)*, May 2001.

[32] V. Poosala. *Histogram-based Estimation Techniques in Databases*. PhD thesis, Univ. of Wisconsin-Madison, 1997.

[33] V. Poosala and V. Ganti. Fast approximate answers to aggregate queries on a data cube. In *Proc. 11th International Conf. on Scientific and Statistical Database Management*, July 1999.

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

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

[36] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 193–204, June 1999.

[37] J. S. Vitter, M. Wang, and B. Iyer. Data cube approximation and histograms via wavelets. In *Proc. 7th International Conf. on Information and Knowledge Management*, pages 96–104, November 1998.

[38] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems*, 15(2):208–229, 1990.