New Sampling-Based Summary Statistics for Improving Approximate Query Answers

Phillip B. Gibbons
Information Sciences Research Center
Bell Laboratories
gibbons@research.bell-labs.com

Yossi Matias
Department of Computer Science
Tel-Aviv University
matias@math.tau.ac.il

Abstract

In large data recording and warehousing environments, it is often advantageous to provide fast, approximate answers to queries, whenever possible. Before DBMSs providing highly-accurate approximate answers can become a reality, many new techniques for summarizing data and for estimating answers from summarized data must be developed. This paper introduces two new sampling-based summary statistics, concise samples and counting samples, and presents new techniques for their fast incremental maintenance regardless of the data distribution. We quantify their advantages over standard sample views in terms of the number of additional sample points for the same view size, and hence in providing more accurate query answers. Finally, we consider their application to providing fast approximate answers to hot list queries. Our algorithms maintain their accuracy in the presence of ongoing insertions to the data warehouse.

1 Introduction

In large data recording and warehousing environments, it is often advantageous to provide fast, approximate answers to queries. The goal is to provide an estimated response in orders of magnitude less time than the time to compute an exact answer, by avoiding or minimizing the number of accesses to the base data.

In a traditional data warehouse set-up, depicted in Figure 1, each query is answered exactly using the data warehouse. We consider instead the set-up depicted in Figure 2, for providing very fast approximate answers to queries. In this set-up, new data being loaded into the data warehouse is also observed by an approximate answer engine. This engine maintains various summary statistics, which we denote *synopsis data structures* or *synopses* [GM97].

Queries are sent to the approximate answer engine. Whenever possible, the engine uses its synopses to promptly return a query response, consisting of an approximate answer and an accuracy measure (e.g., a 95% confidence interval for numerical answers). The user can then decide whether or not to have an exact answer computed from the base data, based on the user's desire for the exact answer and the estimated time for computing an exact answer as determined by the query optimizer and/or the approximate answer

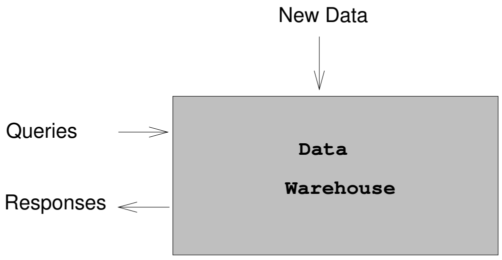


Figure 1: A traditional data warehouse.

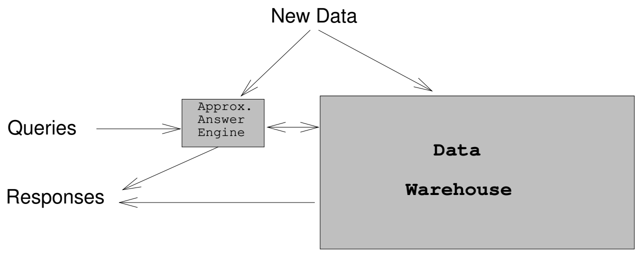


Figure 2: Data warehouse set-up for providing approximate query answers.

engine.[1] There are a number of scenarios for which a user may prefer an approximate answer in a few seconds over an exact answer that requires tens of minutes or more to compute, e.g., during a drill down query sequence in data mining [GM95, HHW97]. Moreover, as discussed by Faloutsos *et al.* [FJS97], sometimes the base data is remote and currently unavailable, so that an exact answer is not an option, until the data again becomes available.

Techniques for fast approximate answers can also be used in a more traditional role within the query optimizer to estimate plan costs, again with very fast response time.

The state-of-the-art in approximate query answers (e.g., [VL93, HHW97, BDF$^+$97]) is quite limited in its speed, scope and accuracy. Before DBMSs providing highly-accurate approximate answers can become a reality, many new techniques for summarizing data and for estimating answers from summarized data must be developed. The goal is to develop effective synopses that capture important information about the data in a concise representation. The important features of the data are determined by the types of queries for which approximate answers are a desirable option. For example, it has been shown that for providing approximate answers

[1] This differs from the *online aggregation* approach in [HHW97], in which the base data is scanned and the approximate answer is updated as the scan proceeds.

to range selectivity queries, the V-optimal histograms capture important features of the data in a concise way [PIHS96].

To handle many base tables and many types of queries, a large number of synopses may be needed. Moreover, for fast response times that avoid disk access altogether, synopses that are frequently used to respond to queries should be memory-resident.[2] Thus we evaluate the effectiveness of a synopsis as a function of its *footprint*, i.e., the number of memory words to store the synopsis. For example, it is common practice to evaluate the effectiveness of a histogram in estimating range selectivities as a function of the histogram footprint (i.e., the number of histogram buckets and the storage requirement for each bucket). Although machines with large main memories are becoming increasingly commonplace, this memory remains a precious resource, as it is needed for query-processing working space (e.g., building hash tables for hash joins) and for caching disk blocks. Moreover, small footprints are more likely to lead to effective use of the processor's L1 and/or L2 cache; e.g., a synopsis that fits entirely in the processor's cache enables even faster response times.

The effectiveness of a synopsis can be measured by the *accuracy* of the answers it provides, and its *response time*. In order to keep a synopsis up-to-date, updates to the data warehouse must be propagated to the synopsis, as discussed above. Thus the final metric is the *update time*.

1.1 Concise samples and counting samples

This paper introduces two new sampling-based summary statistics, *concise samples* and *counting samples*, and presents new techniques for their fast incremental maintenance regardless of the data distribution.

Consider the class of queries that ask for the frequently occurring values for an attribute in a relation of size n. One possible synopsis data structure is the set of attribute values in a uniform random sample of the tuples in the relation: any value occurring frequently in the sample is returned in response to the query. However, note that any value occurring frequently in the sample is a wasteful use of the available space. We can represent k copies of the same value v as the pair $\langle v, k\rangle$, thereby freeing up space for $k-2$ additional sample points.[3] This simple observation leads to our first new sampling-based synopsis data structure:

Definition 1 *A concise sample is a uniform random sample of the data set such that values appearing more than once in the sample are represented as a value and a count.*

While using $\langle$value, count$\rangle$ pairs is common practice in various contexts, we apply it in the context of random samples, such that a concise sample of sample-size m will refer to a sample of $m' \geq m$ sample points whose *concise representation (i.e., footprint) is size* m. This simple idea is quite powerful, and to the best of our knowledge, has never before been studied.

Concise samples are never worse than traditional samples, and can be exponentially or more better depending on the data distribution. We quantify their advantages over traditional samples in terms of the number of additional sample points for the same footprint, and hence in providing more accurate query answers.

Since the number of sample points provided by a concise sample depends on the data distribution, the problem of maintaining a concise sample as new data arrives is more difficult than with ordinary samples. We present a fast algorithm for maintaining a concise sample within a given footprint bound, as new data is inserted into the data warehouse.

Counting samples are a variation on concise samples in which the counts are used to keep track of all occurrences of a value inserted into the relation since the value was selected for the sample. We discuss their relative merits as compared with concise samples, and present a fast algorithm for maintaining counting samples under insertions and deletions to the data warehouse.

In most uses of random samples in estimation, whenever a sample of size n is needed it is extracted from the base data: either the entire relation is scanned to extract the sample, or n random disk blocks must be read (since tuples in a disk block may be highly correlated). With our approximate query set-up, as in [GMP97b], we maintain a random sample at all times. As argued in [GMP97b], maintaining a random sample allows for the sample to be packed into consecutive disk blocks or in consecutive pages of memory. Moreover, for each tuple in the sample, only the attribute(s) of interest are retained, for an even smaller footprint and faster retrieval.

Sampling-based estimation has been shown to be quite useful in the context of query processing and optimization (see, e.g., Chapter 9 in [BDF+97]). The accuracy of sampling-based estimation improves with the size of the sample. Since both concise and counting samples provide more sample points for the same footprint, they provide more accurate estimations.

Note that any algorithm for maintaining a synopsis in the presence of inserts without accessing the base data can also be used to compute the synopsis from scratch in one pass over the data, in limited memory.

1.2 Hot list queries

We consider an application of concise and counting samples to the problem of providing fast (approximate) answers to hot list queries. Specifically, we provide, to a certain accuracy, an ordered set of $\langle$value, count$\rangle$ pairs for the most frequently occurring "values" in a data set, in potentially orders of magnitude smaller footprint than needed to maintain the counts for all values. An example hot list is the top selling items in a database of sales transactions. In various contexts, hot lists of m pairs are denoted as *high-biased histograms* [IC93] of $m+1$ buckets, the first m mode statistics, or the m largest itemsets [AS94]. Hot lists can be maintained on singleton values, pairs of values, triples, etc.; e.g., they can be maintained on k-itemsets for any specified k, and used to produce association rules [AS94, BMUT97]. Hot lists capture the most skewed (i.e., popular) values in a relation, and hence have been shown to be quite useful for estimating predicate selectivities and join sizes (see [Ioa93, IC93, IP95]). In a mapping of values to parallel processors or disks, the most skewed values limit the number of processors or disks for which good load balance can be obtained. Hot lists are also quite useful in data mining contexts for real-time fraud detection in telecommunications traffic [Pre97], and in fact an early version of our algorithm described below has been in use in such contexts for over a year.

Note that the difficulty in incremental maintenance of hot lists is in detecting when itemsets that were small become large due to a shift in the distribution of the newer data. Such detection is difficult since no information is maintained on small itemsets, in order to remain within the footprint bound, and we do not access the base data.

Our solution can be viewed as using a *probabilistic counting scheme* to identify newly-popular itemsets: If τ is the estimated itemset count of the smallest itemset in the hot list, then we add each new item with probability $1/\tau$. Thus, although we cannot afford to maintain counts that will detect when a newly-popular item-

[2] Various synopses can be swapped in and out of memory as needed. For persistence and recovery, combinations of snapshots and/or logs can be stored on disk; alternatively, the synopsis can often be recomputed in one pass over the base data. Such discussions are beyond the scope of this paper.

[3] We assume throughout this paper that values and counts use one "word" of memory each. In general, variable-length encoding could be used for the counts, so that only $\lceil \lg x \rceil$ bits are needed to store x as a count; this reduces the footprint but complicates the memory management.

set has now occurred τ or more times, we probabilistically *expect* to have τ occurrences of the itemset before we (tentatively) add the itemset to the hot list.

We present an algorithm based on concise samples and one based on counting samples. The former has lower overheads but the latter is more accurate. We provide accuracy guarantees for the two methods, and experimental results demonstrating their (often large) advantage over using a traditional random sample. Our algorithms maintain their accuracy in the presence of ongoing insertions to the data warehouse.

This work is part of the Approximate QUery Answering (AQUA) project at Bell Labs. Further details on the Aqua project can be found in [GMP97a, GPA$^+$98].

Outline. Section 2 discusses previous related work. Concise samples are studied in Section 3, and counting samples are studied in Section 4. Finally, in Section 5, we describe their application to hot list queries.

2 Previous related work

Hellerstein, Haas, and Wang [HHW97] proposed a framework for approximate answers of aggregation queries called *online aggregation*, in which the base data is scanned in a random order at query time and the approximate answer for an aggregation query is updated as the scan proceeds. A graphical display depicts the answer and a (decreasing) confidence interval as the scan proceeds, so that the user may stop the process at any time. Our techniques do not provide such continuously-refined approximations; instead we provide a single discrete step of approximation. Moreover, we do not provide special treatment for small sets in group-by operations as outlined by Hellerstein *et al.* Furthermore, since our synopses are precomputed, we must know in advance what are the attribute(s) of interest; online aggregation does not require such advance knowledge (except for its group-by treatment). Finally, we do not consider all types of aggregation queries, and instead study sampling-based summary statistics which can be applied to give sampling-based approximate answers. There are two main advantages of our approach. First is the response time: our approach is many orders of magnitude faster since we provide an approximate answer without accessing the base data. Ours may respond without a single disk access, as compared with the many disk accesses performed by their approach. Second, we do not require that data be read in a random order in order to obtain provable guarantees on the accuracy.

Other systems support limited on-line aggregation features; e.g., the Red Brick system supports running count, average, and sum (see [HHW97]).

There have been several query processors designed to provide approximate answers to set-valued queries (e.g., see [VL93] and the references therein). These operate on the base data at query time and typically define an approximate answer for set-valued queries to be subsets and supersets that converge to the exact answer. There have also been recent works on "fast-first" query processing, whose goal is to quickly provide a few tuples of the query answer. Bayardo and Miranker [BM96] devised techniques for optimizing and executing queries using pipelined, nested-loops joins in order to minimize the latency until the first answer is produced. The Oracle Rdb system [AZ96] provides support for running multiple query plans simultaneously, in order to provide for fast-first query processing.

Barbará *et al.* [BDF$^+$97] presented a survey of *data reduction* techniques, including sampling-based techniques; these can be used for a variety of purposes, including providing approximate query answers. Olken and Rotem [OR92] presented techniques for maintaining random sample views. In [GMP97b], we advocated the use of a *backing sample*, a random sample of a relation that is kept up-to-date, and showed how it can be used for fast incremental maintenance of equi-depth and Compressed histograms. A concise sample could be used as a backing sample, for more sample points for the same footprint.

Matias *et al.* [MVN93, MVY94, MSY96] proposed and studied *approximate data structures* that provide fast approximate answers. These data structures have linear space footprints.

A number of probabilistic techniques have been previously proposed for various counting problems. Morris [Mor78] (see also [Fla85], [HK95]) showed how to approximate the sum of a set of n values in $[1..m]$ using only $O(\lg\lg m + \lg\lg n)$ bits of memory. Flajolet and Martin [FM83, FM85] designed an algorithm for approximating the number of distinct values in a relation in a single pass through the data and using only $O(\lg n)$ bits of memory. Other algorithms for approximating the number of distinct values in a relation include [WVZT90, HNSS95]. Alon, Matias and Szegedy [AMS96] developed sublinear space randomized algorithms for approximating various frequency moments, as well as tight bounds on the minimum possible memory required to approximate such frequency moments. Probabilistic techniques for fast parallel estimation of the size of a set were studied in [Mat92].

None of this previous work considers concise or counting samples.

3 Concise samples

Consider a relation R with n tuples and an attribute A. Our goal is to obtain a uniform random sample of $R.A$, i.e., the values of A for a random subset of the tuples in R.[4]

Since a concise sample represents sample points occurring more than once as $\langle$value, count$\rangle$ pairs, the true sample size may be much larger than its footprint (it is never smaller).

Definition 2 *Let* $S = \{\langle v_1, c_1\rangle, \ldots, \langle v_j, c_j\rangle, v_{j+1}, \ldots, v_\ell\}$ *be a concise sample. Then* sample-size$(S) = \ell - j + \sum_{i=1}^{j} c_i$, *and* footprint$(S) = \ell + j$.

A concise sample S of $R.A$ is a uniform random sample of size sample-size(S), and hence can be used as a uniform random sample in any sampling-based technique for providing approximate query answers.

Note that if there are at most $m/2$ distinct values for $R.A$, then a concise sample of sample-size n has a footprint at most m (i.e., in this case, the concise sample is the exact histogram of $\langle$value, count$\rangle$ pairs for $R.A$). Thus, the sample-size of a concise sample may be arbitrarily larger than its footprint:

Lemma 1 *For any footprint $m \geq 2$, there exists data sets for which the sample-size of a concise sample is n/m times larger than its footprint, where n is the size of the data set.*

Since the sample-size of a traditional sample equals its footprint, Lemma 1 implies that for such data sets, the concise sample has n/m times as many sample points as a traditional sample of the same footprint.

Offline/static computation. We first describe an algorithm for extracting a concise sample of footprint m from a static relation residing on disk. First, repeat m times: select a random tuple from the relation (this typically takes multiple disk reads per tuple [OR89, Ant92]) and extract its value for attribute A. Next, semi-sort the set of values, and replace every value occurring multiple times with a $\langle$value, count$\rangle$ pair. Then, continue to sample until either adding the sample point would increase the concise sample

[4] For simplicity, we describe our algorithms here and in the remainder of the paper in terms of a single attribute, although the approaches apply equally well to pairs of attributes, etc.

footprint to $m + 1$ (in which case this last attribute value is ignored) or n samples have been taken. For each new value sampled, look-up to see if it is already in the concise sample and then either add a new singleton value, convert a singleton to a $\langle$value, count$\rangle$ pair, or increment the count for a pair. To minimize the cost, sample points can be taken in batches and stored temporarily in the working space memory and a look-up hash table can be constructed to enable constant-time look-ups; once the concise sample is constructed, only the concise sample itself is retained. If m' sample points are selected in all (i.e., the sample-size is m'), the cost is $\Theta(m')$ disk accesses. The incremental approach we describe next requires no disk accesses, given the set-up depicted in Figure 2. In general, it can also be used to compute a concise sample in one sequential pass over a relation.

3.1 Incremental maintenance of concise samples

We present a fast algorithm for maintaining a concise sample within a given footprint bound as new data is inserted into the data warehouse. Since the number of sample points provided by a concise sample depends on the data distribution, the problem of maintaining a concise sample as new data arrives is more difficult than with traditional samples. The reservoir sampling algorithm of Vitter [Vit85], that can be used to maintain a traditional sample in the presence of insertions of new data (see [GMP97b] for extensions to handle deletions), relies heavily on the fact that we know in advance the sample-size (which, for traditional samples, equals the footprint size). With concise samples, the sample-size depends on the data distribution to date, and any changes in the data distribution must be reflected in the sampling frequency.

Our maintenance algorithm is as follows. We set up an entry threshold τ (initially 1) for new tuples to be selected for the sample. Let S be the current concise sample and consider a new tuple t. With probability $1/\tau$, we add $t.A$ to S. We do a look-up on $t.A$ in S. If it is represented by a pair, we increment its count. Otherwise, if $t.A$ is a singleton in S, we create a pair, or if it is not in S, we create a singleton. In these latter two cases we have increased the footprint by 1, so if the footprint for S was already equal to the prespecified footprint bound, then we need to evict existing sample points to create room.

In order to create room, we raise the threshold to some τ' and then subject each sample point in S to this higher threshold. Specifically, each of the sample-size(S) sample points is evicted with probability τ/τ'. We expect to have sample-size(S) $\cdot$ (τ/τ') sample points evicted. Note that the footprint is only decreased when a $\langle$value, count$\rangle$ pair reverts to a singleton or when a value is removed altogether. If the footprint has not decreased, we raise the threshold and try again. Subsequent inserts are selected for the sample with probability $1/\tau'$.

Theorem 2 *For any sequence of insertions, the above algorithm maintains a concise sample.*

Proof. Let τ be the current threshold. We maintain the invariant that each tuple in the relation has been treated as if the threshold were always τ. The crux of the proof is to show that this invariant is maintained when the threshold is raised to τ'. Each of the sample-size(S) sample points is evicted with probability τ/τ'. If it was not in S prior to creating room, then by the inductive invariant, a coin with heads probability $1/\tau$ was flipped and failed to come up heads for this tuple. Thus the same probabilistic event would fail to come up heads with the new, stricter coin (with heads probability only $1/\tau'$). If it was in S prior to creating room, then by the inductive invariant, a coin with heads probability $1/\tau$ came up heads. Since $(1/\tau) \cdot (\tau/\tau') = (1/\tau')$, the result is that the tuple is in the sample with probability $1/\tau'$. Thus the inductive invariant is indeed maintained. ∎

The algorithm maintains a concise sample regardless of the sequence of increasing thresholds used. Thus, there is complete flexibility in deciding, when raising the threshold, what the new threshold should be. A large raise may evict more than is needed to reduce the sample footprint below its upper bound, resulting in a smaller sample-size than there would be if the sample footprint matches the upper bound. On the other hand, evicting more than is needed creates room for subsequent additions to the concise sample, so the procedure for creating room runs less frequently. A small raise also increases the likelihood that the footprint will not decrease at all, and the procedure will need to be repeated with a higher threshold.

For simplicity in the experiments reported in Section 3.3, we raised the threshold by 10% each time. Note that in general, one can improve threshold selection at a cost of a more elaborate algorithm, e.g., by using binary search to find a threshold that will create the desired decrease in the footprint or by setting the threshold so that $(1 - \tau/\tau')$ times the number of singletons is a lower bound on the desired decrease in the footprint.

Note that instead of flipping a coin for each insert into the data warehouse, we can flip a coin that determines how many such inserts can be skipped before the next insert that must be placed in the sample (as in Vitter's reservoir sampling Algorithm X [Vit85]): the probability of skipping over exactly i elements is $(1-1/\tau)^i \cdot (1/\tau)$. As τ gets large, this results in a significant savings in the number of coin flips and hence the update time. Likewise, since the probability of evicting a sample point is typically small (i.e., τ'/τ is a small constant), we can save on coin flips and decrease the update time by using a similar approach when evicting.

Raising a threshold costs $O(m')$, where m' is the sample-size of the concise sample before the threshold was raised. For the case where the threshold is raised by a constant factor each time, we expect there to be a constant number of coin tosses resulting in sample points being retained for each sample point evicted. Thus we can amortize the retained against the evicted, and we can amortize the evicted against their insertion into the sample (each sample point is evicted only once). It follows that even taking into account the time for each threshold raise, we have an $O(1)$ amortized expected update time per insert, regardless of the data distribution.

3.2 Quantifying the sample-size advantage of concise samples

The expected sample-size increases with the skew in the data. By Lemma 1, the advantage is unbounded for certain distributions. We show next that for exponential distributions, the advantage is exponential:

Theorem 3 *Consider the family of exponential distributions: for* $i = 1, 2, \cdots$, $\Pr(v = i) = \alpha^{-i}(\alpha - 1)$, *for* $\alpha > 1$. *For any footprint* $m \geq 2$, *the expected sample-size of a concise sample with footprint* m *is at least* $\alpha^{m/2}$.

Proof. The expected sample-size can be lower bounded by the expected number of randomly selected tuples before the first tuple whose attribute value v is greater than $m/2$. (When all values are at most $m/2$ then we can fit each value and its count, if any, within the footprint.) The probability of selecting a value greater than $m/2$ is

$$\sum_{i=m/2+1}^{\infty} \alpha^{-i}(\alpha - 1) = \alpha^{-m/2} \ ,$$

so the expected number of tuples selected before such an event occurs is $\alpha^{m/2}$. ∎

Next, we evaluate the expected gain in using a concise sample over a traditional sample for arbitrary data sets. The estimate is given in terms of the frequency moment F_k, for $k \geq 2$, of the data set, defined as $F_k = \sum_j n_j^k$, where j is taken over the values represented in the set and n_j is the number of set elements of value j.

Theorem 4 *For any data set, when using a concise sample S with sample-size m, the expected gain is*

$$E[m - \textit{number of distinct values in } S] = \sum_{k=2}^{m} (-1)^k \binom{m}{k} \frac{F_k}{n^k} .$$

Proof. Let $p_j = n_j/n$ be the probability that an item selected at random from the set is of value j. Let X_i be an indicator random variable so that $X_i = 1$ if the ith item selected to be in the traditional sample has a value not represented as yet in the sample, and $X_i = 0$ otherwise. Then, $\Pr(X_i = 1) = \sum_j p_j (1-p_j)^{i-1}$, where j is taken over the values represented in the set (since $X_i = 1$ if some value j is selected so that it has not been selected in any of the first $i-1$ steps). Clearly, $X = \sum_{i=1}^m X_i$ is the number of distinct values in the traditional sample. We can now evaluate E[number of distinct values] as

$$\begin{aligned} E[X] &= \sum_{i=1}^{m} E[X_i] = \sum_{i=1}^{m} \sum_j p_j (1-p_j)^{i-1} \\ &= \sum_j p_j \frac{1-(1-p_j)^m}{1-(1-p_j)} = \sum_j (1-(1-p_j)^m) \\ &= \sum_j \left(1 - \sum_{k=0}^{m} (-1)^k \binom{m}{k} p_j^k \right) \\ &= \sum_{k=1}^{m} (-1)^{k+1} \binom{m}{k} \frac{F_k}{n^k} . \end{aligned}$$

∎

Note that the footprint for a concise sample is at most twice the number of distinct values.

3.3 Experimental evaluation

We conducted a number of experiments evaluating the gain in the sample-size of concise samples over traditional samples. In each experiment, 500K new values were inserted into an initially empty data warehouse. Since the exact attribute values do not effect the relative quality of our techniques, we chose the integer value domain from $[1, D]$, where D, the potential number of distinct values, was varied from 500 to 50K. We used a large variety of Zipf data distributions. The zipf parameter was varied from 0 to 3 in increments of 0.25; this varies the skew from nonexistent (the case of zipf parameter = 0 is the uniform distribution) to quite large. Most of the experiments restricted each sample to footprint $m = 1000$. However, to stress the algorithms, we also considered footprint $m = 100$. Recall that if the ratio D/m is $\leq .5$, then all values inserted into the warehouse can be maintained in the concise sample. We consider $D/m = 5$, 50, and 500. Each data point plotted is the average of 5 trials.

Each experiment compares the sample-size of the samples produced by three algorithms, with the same footprint m.

- *traditional:* a random sample of size m is maintained using reservoir sampling.
- *concise online:* the algorithm described in Section 3.1.
- *concise offline:* the offline/static algorithm described at the beginning of Section 3.

The offline algorithm is plotted to show the intrinsic sample-size of concise samples for the given distribution. The gap between the online and the offline is the penalty our online algorithm pays in terms of loss in sample-size due to suboptimal adjustments in the threshold. In the experiments plotted, whenever the threshold is raised, the new threshold is set to $\lceil 1.1 \times \tau \rceil$, where τ is the current threshold.

Figure 3 depicts the sample-size as a function of the zipf parameter for varying footprints and D/m ratios. First, in (a) and (b), we compare footprint 100 and 1000, respectively, for the same data sets. The sample-size for traditional samples, which equals the footprint, is so small that it is hidden by the x-axis in these plots. At the scale shown in these two plots, the other experiments we performed for footprint 100 and 1000 gave nearly identical results. These results show that for high skew the sample-size for concise samples grows up to 3 orders of magnitude larger than for traditional samples, as anticipated. Also, the online algorithm is within 15% of the offline algorithm for footprint 1000 and within 28% when constrained to use only footprint 100.

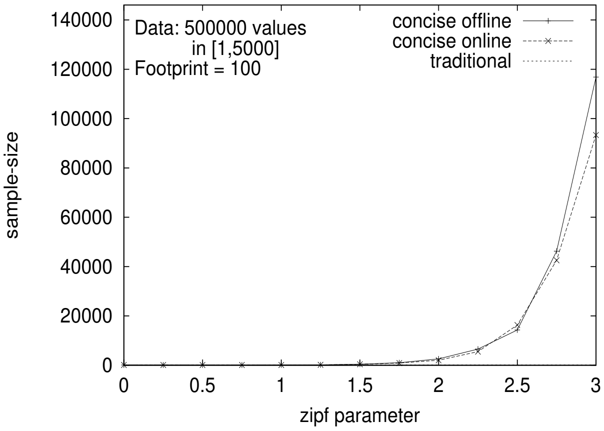
Second, in (c) and (d), we show representative plots of our experiments depicting how the gain in sample-size is effected by the D/m ratio. In these plots, we compare $D/m = 50$ and $D/m = 5$, respectively, for the same footprint 1000. We have truncated these plots at zipf parameter 1.5, to permit a more closer examination of the sample-size gains for zipf parameters near 1.0. (In fact, Figure 3(d) is simply a more detailed look at the data points in Figure 3(b) up to zipf parameter 1.5.)

Recall that for $D/m = .5$, the sample-size for concise samples is a factor of n/m larger than that for traditional samples, regardless of the zipf parameter. These figures show that for $D/m = 5$, there are no noticeable gains in sample-size for concise samples until the zipf parameter is > 0.5, and for $D/m = 50$, there are no noticeable gains until the zipf parameter is > 0.75. The improvements with smaller D/m arise since m/D is the fraction of the distinct values for which counts can be maintained.

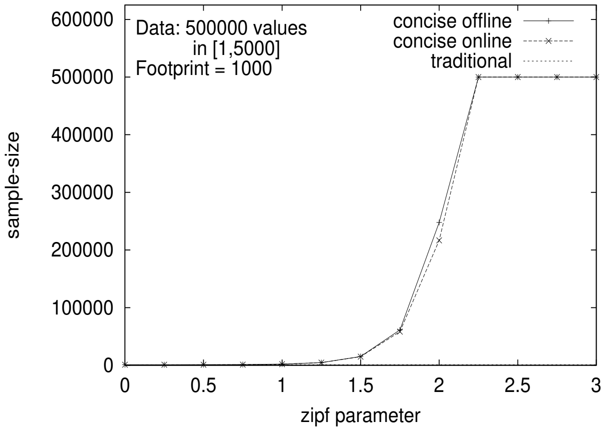
Update time overheads. There are two main sources of update time overheads associated with our (online) concise sampling algorithm. First, there are the coin flips that must be performed to decide which inserts are added to the concise sample and to evict values from the concise sample when the threshold is raised. Recall that we use the technique in [Vit85] that minimizes the number of coin flips by computing, for a given coin bias, how many flips of the coin until the next heads (or next tails, depending on which type of flip requires an action to be performed by the algorithm). Since the algorithm does work only when we have such a coin flip, the number of coin flips is a good measure of the update time overheads. For each of the data distribution and footprint scenarios presented in Figure 3, we report in Table 1 the average coin flips for each new insert to the data warehouse.

Second, there are the lookups into the current concise sample to see if a value is already present in the sample. The coin flip measure does not account for the work done in initially populating the concise sample: on start-up, the algorithm places every insert into the concise sample until it has exceeded its footprint. A lookup is performed for each of these, so the lookup measure accounts for this cost, as well as the lookups done when an insert is selected for the concise sample due to a coin flip. For each of the data distribution and footprint scenarios presented in Figure 3, we report in Table 1 the number of lookups per insert to the data warehouse.

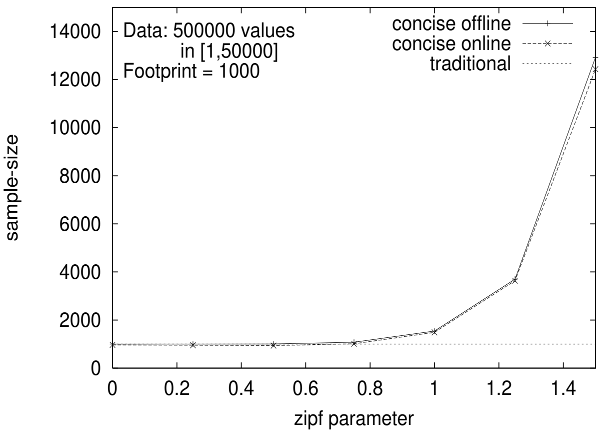
As can be seen from the table, the overheads are quite small. The overheads are smallest for small zipf parameters. There is very little dependence on the D/m ratio. An order of magnitude decrease in the footprint results in roughly an order of magnitude



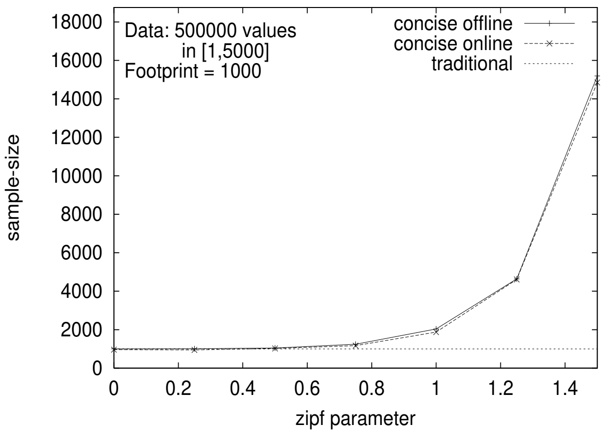
(a)



(b)



(c)



(d)

Figure 3: Comparing sample-sizes of concise and traditional samples as a function of skew, for varying footprints and D/m ratios. In (a) and (b), we compare footprint 100 and footprint 1000, respectively, for the same data sets. In (c) and (d), we compare $D/m = 50$ and $D/m = 5$, respectively, for the same footprint 1000.

Table 1: Coin flips and lookups per insert for the experiments in Figure 3. These are abstract measures of the computation costs: the number of instructions executed by the algorithm is directly proportional to the number of coin flips and lookups, and is dominated by these two factors.

zipf param	Fig. 3(a) flips	Fig. 3(a) lookups	Figs. 3(b)(d) flips	Figs. 3(b)(d) lookups	Fig. 3(c) flips	Fig. 3(c) lookups
0.00	0.003	0.002	0.023	0.013	0.023	0.013
0.25	0.003	0.002	0.023	0.013	0.023	0.013
0.50	0.003	0.002	0.024	0.014	0.023	0.013
0.75	0.003	0.002	0.027	0.016	0.024	0.014
1.00	0.004	0.002	0.041	0.024	0.032	0.019
1.25	0.006	0.003	0.079	0.049	0.066	0.040
1.50	0.011	0.007	0.188	0.124	0.170	0.111
1.75	0.023	0.013	0.426	0.333	0.406	0.306
2.00	0.045	0.027	0.559	0.744	0.645	0.726
2.25	0.097	0.061	0.000	1.000	0.000	1.000
2.50	0.189	0.125	0.000	1.000	0.000	1.000
2.75	0.363	0.271	0.000	1.000	0.000	1.000
3.00	0.544	0.482	0.000	1.000	0.000	1.000

decrease in the overheads for zipf parameters below 2. For zipf parameters above 2, all values fit within the footprint 1000, so there is exactly one lookup and zero coin flips per insert to the data warehouse. Each of these results can be understood by observing that for a given threshold, the expected number of flips and lookups is inversely proportional to the threshold. Moreover, the expectation of the sample-size is equal to the number of inserts divided by the current threshold. Thus the flips and lookups per insert increases with increasing sample-size (except when the flips drop to zero as discussed above).

Note that despite the procedure to revisit sample points and perform coin flips whenever the threshold is raised, the number of flips per insert is at worst 0.645, and often orders of magnitude smaller. This is due to a combination of two factors: if the threshold is raised a large amount, then the procedure is done less often, and if it is raised only a small amount, then very few flips are needed in the procedure (since we are using [Vit85]).

4 Counting samples

In this section, we define counting samples, present an algorithm for their incremental maintenance, and provide analytical guarantees on their performance.

Consider a relation R with n tuples and an attribute A. Counting samples are a variation on concise samples in which the counts are used to keep track of all occurrences of a value inserted into the relation since the value was selected for the sample.[5] Their definition is motivated by a sampling&counting process of this type from a static data warehouse:

Definition 3 *A counting sample for $R.A$ with threshold τ is any subset of $R.A$ obtained as follows:*

1. *For each value v occurring $c > 0$ times in R, we flip a coin with probability $1/\tau$ of heads until the first heads, up to at most c coin tosses in all; if the ith coin toss is heads, then v occurs $c-i+1$ times in the subset, else v is not in the subset.*
2. *Each value v occurring $c > 1$ times in the subset is represented as a pair $\langle v, c\rangle$, and each value v occurring exactly once is represented as a singleton v.*

[5] In other words, since we have set aside a memory word for a count, why not count the subsequent occurrences exactly?

Obtaining a concise sample from a counting sample. Although counting samples are not uniform random samples of the base data, they can be used to obtain such a sample without any further access to the base data. Specifically, a concise sample can be obtained from a counting sample by considering each pair $\langle v, c\rangle$ in the counting sample in turn, and flipping a coin with probability $1/\tau$ of heads $c-1$ times and reducing the count by the number of tails. The footprint decreases by one for each pair for which all its coins are tails.

4.1 Incremental maintenance of counting samples

Our incremental maintenance algorithm is as follows. We set up an entry threshold τ (initially 1) for new tuples to be selected for the counting sample. Let S be the current counting sample and consider a new tuple t. We do a look-up on $t.A$ in S. If $t.A$ is represented by a ⟨value, count⟩ pair in S, we increment its count. If $t.A$ is a singleton in S, we create a pair. Otherwise, $t.A$ is not in S and we add it to S with probability $1/\tau$.

If the footprint for S now exceeds the prespecified footprint bound, then we need to evict existing values to create room. As with concise samples, we raise the threshold to some τ', and then subject each value in S to this higher threshold. The process is slightly different for counting samples, since the counts are different.

For each value in the counting sample, we flip a biased coin, decrementing its observed count on each flip of tails until either the count reaches zero or a heads is flipped. The first coin toss has probability of heads τ/τ', and each subsequent coin toss has probability of heads $1/\tau'$. Values with count zero are removed from the counting sample; other values remain in the counting sample with their (typically reduced) counts. (The overall number of coin tosses can be reduced to a constant per value using an approach similar to that described for concise samples, since we stop at the first heads (if any) for each value.) Thus raising a threshold costs $O(m)$, where m is the number of distinct values in the counting sample (which is at most the footprint). If the threshold is raised such a constant factor each time, we expect there to be a constant number of sample points removed for each sample point flipping a heads. Thus as in concise sampling, it follows that we have a constant amortized expected update time per data warehouse insert, regardless of the data distribution.

An advantage of counting samples over concise samples is that we can maintain counting samples in the presence of deletions to the data warehouse. Maintaining concise samples in the presence of such deletions is difficult: If we fail to delete a sample point in response to the delete operation, then we risk having the sample fail to be a subset of the data set. On the other hand, if we always delete a sample point, then the sample may no longer be a random sample of the data set. With counting samples, we do not have this difficulty. For a delete of a value v, we look-up to see if v is in the counting sample (using a hash function), and decrement its count if it is. Thus we have $O(1)$ expected update time for deletions to the data warehouse.

Theorem 5 *For any sequence of insertions and deletions, the above algorithm maintains a counting sample.*

Proof. We must show that properties 1 and 2 of the definition of a counting sample are preserved when an insert occurs, a delete occurs, or the threshold is raised.

An insert of a value v increases by one its count in R. If the value is in the counting sample, then one of its coin flips was heads, and we increment the count in the counting sample. Otherwise, none of its coin flips to date were heads, and the algorithm flips a coin with the appropriate probability. All other values are untouched, so property 1 is preserved.

A delete of a value v decreases by one its count in R. If the value is in the counting sample, then the algorithm decrements the count (which may drop the count to 0). Otherwise, c coin flips occurred to date and were tails, so the first $c-1$ were also tails, and the value remains omitted from the counting sample. All other values are untouched, so property 1 is preserved.

Consider raising the threshold from τ to τ', and let v be a value occurring $c > 0$ times in R. If v is not in the counting sample, there were c coin flips with heads probability $1/\tau$ that came up tails. Thus the same c probabilistic events would fail to come up heads with the new, stricter coin (with heads probability only $1/\tau'$). If v is in the counting sample with count c', then there were $c - c'$ coin flips with heads probability $1/\tau$ that came up tails, and these same probabilistic events would come up tails with the stricter coin. This was followed by a coin flip with heads probability $1/\tau$ that came up heads, and the algorithm flips a coin with heads probability τ/τ', so that the result is the same as a coin flip with probability $(1/\tau) \cdot (\tau/\tau') = (1/\tau')$. If this coin comes up tails, then subsequent coin flips for this value have heads probability $1/\tau'$. In this way, property 1 is preserved for all values.

In all cases, property 2 is immediate, and the theorem is proved. ∎

Note that although both concise samples and counting samples have $O(1)$ amortized update times, counting samples are slower to update than concise samples, since, unlike concise sample, they perform a look-up (into the counting sample) at each update to the data warehouse.

Theorem 6 *Let R be an arbitrary relation, and let τ be the current threshold for a counting sample S. (i) Any value v that occurs at least τ times in R is expected to be in S. (ii) Any value v that occurs f_v times in R will be in S with probability $1 - (1 - \frac{1}{\tau})^{f_v}$. (iii) For all $\alpha > 1$, if $f_v \geq \alpha \cdot \tau$, then with probability $\geq 1 - e^{-\alpha}$, the value will be in S and its count will be at least $f_v - \alpha\tau$.*

Proof. Claims (i) and (ii) follow immediately from property 1 of counting samples. As for (iii), $\Pr(v \in S \text{ with count } \geq f_v - \alpha\tau) = 1 - \Pr(\text{the first } \alpha\tau \text{ coin tosses for } v \text{ are all tails}) = 1 - (1 - \frac{1}{\tau})^{\alpha\tau}$ $\geq 1 - e^{-\alpha}$. ∎

5 Hot list queries

In this section, we present new algorithms for providing approximate answers to hot list queries. Recall that hot list queries request an ordered set of ⟨value, count⟩ pairs for the k most frequently occurring data values, for some k.

5.1 Algorithms

We present four algorithms for providing fast approximate answers to hot list queries for a relation R with n tuples, based on incrementally maintained synopses with footprint bound m, $m \geq 2k$.

Using traditional samples. A traditional sample of size m can be maintained using Vitter's reservoir sampling algorithm [Vit85]. To report an approximate hot list, we first semi-sort by value, and replace every sample point occurring multiple times by a ⟨value, count⟩ pair. We then compute the k'th largest count c_k, and report all pairs with counts at least $\max(c_k, \delta)$, scaling the counts by n/m, where δ is a confidence threshold (discussed below). Note that there may be fewer than k distinct values in the sample, so fewer than k pairs may be reported (even when using the minimal confidence threshold $\delta = 1$). The response time for reporting is $O(m)$.

Using concise samples. A concise sample of footprint m can be maintained using the algorithm of Section 3. To report an approximate hot list, we first compute the k'th largest count c_k (using a linear time selection algorithm). We report all pairs with counts at least $\max(c_k, \delta)$, scaling the counts by n/m', where δ is a confidence threshold and m' is the sample-size of the concise sample. Note that when $\delta = 1$, we will report k pairs, but with larger δ, fewer than k may be reported. The response time for reporting is $O(m)$. Alternatively, we can trade-off update time vs. response time by keeping the concise sample sorted by counts. This allows for reporting in $O(k)$ time.

Using counting samples. A counting sample of footprint m can be maintained using the algorithm of Section 4. To report an approximate hot list, we use the same algorithm as described above for using concise samples, except that instead of scaling the counts, we add to the counts a compensation, $\hat{c}$, determined by the analysis below. This augmentation of the counts serves to compensate for inserts of a value into the data warehouse prior to the successful coin toss that placed it in the counting sample. Let τ be the current threshold. We report all pairs with counts at least $\max(c_k, \tau - \hat{c})$. Given the conversion of counting samples into concise samples discussed in Section 4, this can be seen to be similar to taking $\delta = 2 - \frac{\hat{c}+1}{\tau}$. (Using the value of $\hat{c}$ determined below, $\delta = 1.582$.)

Full histogram on disk. The last algorithm maintains a full histogram on disk, i.e., ⟨value, count⟩ pairs for all distinct values in R, with a copy of the top $m/2$ pairs stored as a synopsis within the approximate answer engine. This enables exact answers to hot list queries. The main drawback of this approach is that each update to R requires a separate disk access to update the histogram. Moreover, it incurs a (typically large) disk footprint that may be on the order of n. Thus this approach is considered only as a baseline for our accuracy comparisons.

5.2 Analysis

The confidence threshold δ. The threshold δ is used to bound the error. The larger the δ, the greater the probability that for reported values, the counts are quite accurate. On the other hand, the larger the δ the greater the probability that fewer than k pairs will be reported. For its use with traditional samples and concise samples, δ must be an integer (unlike with counting samples, where it need not be). We have found that $\delta = 3$ is a good choice, and use that value in our experiments in Section 5.3.

To study the effect of δ on the accuracy, we consider in what follows hot list queries of the form "report all pairs that can be reported with confidence". That is, we report all values occurring at least δ times in the traditional or concise sample. The accuracy of the approximate hot list reported using concise sampling is summarized in the following theorem:

Theorem 7 *Let R be an arbitrary relation of size n, and let τ be the current threshold for a concise sample S. Then:*

1. Frequent values will be reported: *For any ϵ, $0 < \epsilon < 1$, any value v with $f_v \geq \tau\delta/(1-\epsilon)$ will be reported with probability at least $1 - e^{-\delta\epsilon^2/(2(1-\epsilon))}$. As an example, when $\epsilon = 1/2$, the reporting probability is $1 - e^{-\delta/4}$.*

2. Infrequent values will not be reported: *For any ϵ, $0 < \epsilon < 1$, any value v with $f_v \leq \tau\delta/(1+\epsilon)$ will be reported with probability less than $e^{-\delta\epsilon^2/(3(1+\epsilon))}$. As an example, when $\epsilon \approx 1$, the (false) reporting probability is less than $e^{-\delta/6}$.*

Proof. These are shown by first reducing the problem to the case where the threshold has always been τ, and then applying a straightforward analysis using Chernoff bounds. ∎

Determination of $\hat{c}$. The value of $\hat{c}$, used in reporting approximate hot lists using counting samples, is determined analytically as follows. Consider a value v in the counting sample S, with count c_v, and let f_v be the number of times the value occurs in R. Let $\text{Est}_v = c_v + \hat{c}$. We will select $\hat{c}$ so that Est_v will be close to f_v. In particular, we want $\mathbf{E}\left(\text{Est}_v | v \text{ is in } S\right) = f_v$. We have that $\mathbf{E}\left(\text{Est}_v | v \text{ is in } S\right) = \hat{c} + \sum_{i=1}^{f_v}(f_v - i + 1) \cdot \Pr(v \text{ was inserted at the } i\text{th occurrence} \mid v \text{ is in } S)$ which after a lengthy calculation equals $\hat{c} + f_v - \tau + 1 + \frac{f_v q^{f_v}}{1-q^{f_v}}$, where $q = 1 - 1/\tau$. Thus we need $\hat{c} \approx \tau - 1 - f_v \frac{(1/e)^{f_v/\tau}}{1-(1/e)^{f_v/\tau}} = \tau - 1 - \frac{f_v}{e^{f_v/\tau}-1}$. Since $\hat{c}$ depends on f_v, which we do not know, we select $\hat{c}$ so as to compensate exactly when $f_v = \tau$ (in this way, $\hat{c}$ is the most accurate when it matters most: smaller f_v should not be reported and the value of $\hat{c}$ is less important for larger f_v). Thus $\hat{c} = \tau\left(1 - \frac{1}{e-1}\right) - 1 = \tau\left(\frac{e-2}{e-1}\right) - 1 \approx .418 \cdot \tau - 1$.

Theorem 8 *Let R be an arbitrary relation, and let τ be the current threshold for a counting sample S. (i) Any value v that occurs $f_v < .582 \cdot \tau$ times in R will not be reported. (ii) For all $\alpha > 1$, any value v that occurs $f_v \geq \alpha \cdot \tau$ times in R, will be reported with probability $\geq 1 - e^{-(\alpha-.582)}$. (iii) If v is in S, its augmented count will be in $[f_v - \beta \cdot \tau\ ,\ f_v + .418 \cdot \tau - 1]$ with probability $\geq 1 - e^{-(\beta+.418)}$, for all $\beta > 0$.*

Proof. The algorithm will fail to report v if its count is less than $\tau - \hat{c}$, i.e. count $\leq .582\tau$. Claim (i) follows. For the case where $f_v \geq \alpha \cdot \tau$, we have count $\leq .582\tau$ if the first $f_v - .582\tau$ coin tosses are all tails, which happens with probability $(1-\frac{1}{\tau})^{f_v-.582\tau}$, which is less than $e^{-(f_v/\tau-.582)} \leq e^{-(\alpha-.582)}$. Claim (ii) follows. The augmented count is at most $f_v + \hat{c}$. It is less than $f_v - \beta \cdot \tau$ if the unaugmented count is at most $f_v - (\beta + .418)\tau$ which happens if the first $(\beta + .418)\tau$ coin tosses are all tails, which happens with probability $< e^{-(\beta+.418)}$. Claim (iii) follows. ∎

On reporting fewer than k values. Our algorithms report fewer than k values for certain data distributions. Alon *et al.* [AMS96] showed that any randomized online algorithm for approximating the frequency of the mode of a given data set to within a constant factor (with probability $> 1/2$) requires space *linear* in the number of distinct values D. This implies that even for $k = 1$, *any* algorithm for answering approximate hot list queries based on a synopsis whose footprint is sublinear in D will fail to be accurate for certain data distributions. Thus in order to report only highly-accurate answers, it is inevitable that fewer than k values are reported for certain distributions.

Note that the problematic data distributions are the nearly-uniform ones with relatively small maximum frequency (this is the case in which the lower bound of Alon *et al.* is proved). Fortunately, it is the skewed distributions, not the nearly-uniform ones, that are of interest, and the algorithms report good results for skewed distributions.

5.3 Experimental evaluation

We conducted a number of experiments comparing the accuracy and overheads of the algorithms for approximate hot lists described in Section 5.1. In each experiment, 500K new values were inserted into an initially empty data warehouse. Since the exact attribute values do not effect the relative quality of the techniques, we chose the integer value domain from $[1, D]$, where D was varied from 500 to 50K. We used a variety of Zipf data distributions, focusing on the modest skew cases where the zipf parameter is 1.0, 1.25, or 1.5. Each of the three approximation algorithms are provided the same footprint m. Most of the experiments studied the footprint $m = 1000$ case. However, to stress the algorithms, we also considered footprint $m = 100$. Recall that if the ratio D/m is $\leq .5$, then all values inserted into the warehouse can be maintained in both the concise sample and the counting sample. As before, we consider $D/m = 5$, 50, and 500.

Only the points reported by each algorithm are plotted. For the algorithms using traditional samples or concise samples, we use a confidence threshold $\delta = 3$. Whenever the threshold is raised, the new threshold is set to $\lceil 1.1 \times \tau \rceil$, where τ is the current threshold. These values gave better results than other choices we tried.

For the following explanation of the plots, we refer the reader to Figure 4. This plots the most frequent values in the data warehouse in order of nonincreasing counts, together with their counts. The x-axis depicts the rank of a value (the actual values are irrelevant here); the y-axis depicts the count for the value with that rank. The k most frequent values are plotted, where k is the number of values whose frequency matches or exceeds the minimum reported count over the three approximation algorithms. Also plotted are values reported by one or more of the approximation algorithms that do not belong among the k most frequent values (to show false positives). These values are tacked on at the right (after the short vertical line below the x-axis, e.g., between 22 and 23 in this figure) in nonincreasing order of their actual frequency; the x-axis typically will not equal their rank since unreported values are not plotted, creating gaps in the ranks. The exact counts are plotted as histogram boxes.

The values and (estimated) counts reported by the three approximation algorithms are plotted, one point per value reported. Any gap in the values reported by an algorithm represents a false negative. For example, using traditional samples has false negatives for the values with rank 7 and 8. The difference between a reported count and the top of the histogram box is the error in the reported count.

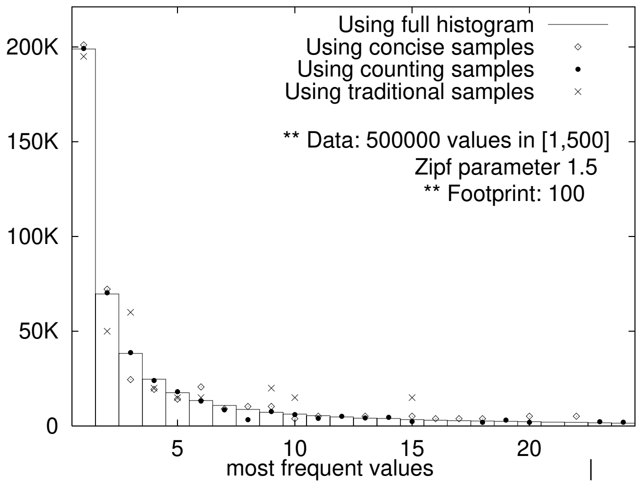


Figure 4: Comparison of hot-list algorithms, depicting the frequency of the most frequent values as reported by the four algorithms.

Figure 4 shows that even with a small footprint, good results are obtained by the algorithms using concise samples and using counting samples. Specifically, using counting samples accurately reported the 15 most frequent values, 18 of the first 20, and had only two false positives (both of which were reported with a $\leq$ 37% overestimation in the counts). The count of the most frequent value was accurate to within .14%. Likewise, using concise samples did almost as well as using counting samples, and much better than using traditional samples. Using concise samples achieves better results than using traditional samples because the sample-size was over 3.8 times larger. Using counting samples achieves better results than using concise samples because the error in the counts is only a one-time error arising prior to a value's last tails flip with the final threshold.

In order to depict plots from our experiments with footprint 1000, we needed to truncate the y-axis to improve readability. All three approximation algorithms perform quite well at the handful of (the most frequent) values not shown due to this truncation[6], so it is more revealing to focus on the rest of the plot.

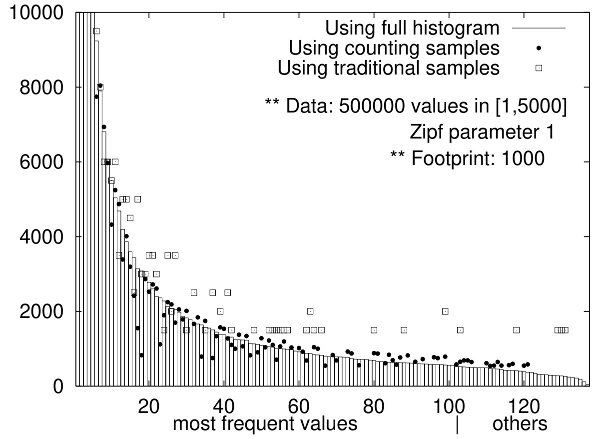


Figure 5: Counting vs. traditional on a less skewed distribution (zipf parameter 1.0), using footprint 1000.

Figure 5 compares using counting samples versus using traditional samples on a less skewed distribution (zipf parameter equals 1.0). With a traditional sample of size 1000, there are only a handful of possible counts that can be reported, with each increment in the number of sample points for a value adding 500 to the reported count. This explains the horizontal rows of reported counts in the figure. As in the previous plot, using counting samples performed quite well, using concise samples (not shown to avoid cluttering the plot) performed not quite as well, and using traditional samples performed significantly worse.

Finally, in Figure 6, we plot the accuracy of the three approximation algorithms on an intermediate skewed distribution (zipf parameter equals 1.25). This plot also depicts the case of a larger D/m ratio than the previous two plots. For readability, each algorithm has its own plot, and the histogram boxes for the exact counts have been replaced with a line connecting these counts. As above, using counting samples is more accurate than using concise samples which is more accurate than using traditional samples. The concise sample-size is nearly 3.5 times larger than the traditional sample-size, leading to the differences between them shown in the plots.

Table 2 reports on the overheads of each approximation algorithm in terms of the number of coin flips and the number of lookups for each new insert to the data warehouse. By these metrics, using traditional samples is better than using concise samples is better than using counting samples, as anticipated. Also shown

[6]For example, in Figure 5, the reported counts for truncated values using concise samples had 5%–16% error, using counting samples had 1%–4% error, and using traditional samples had 3%–31% error.

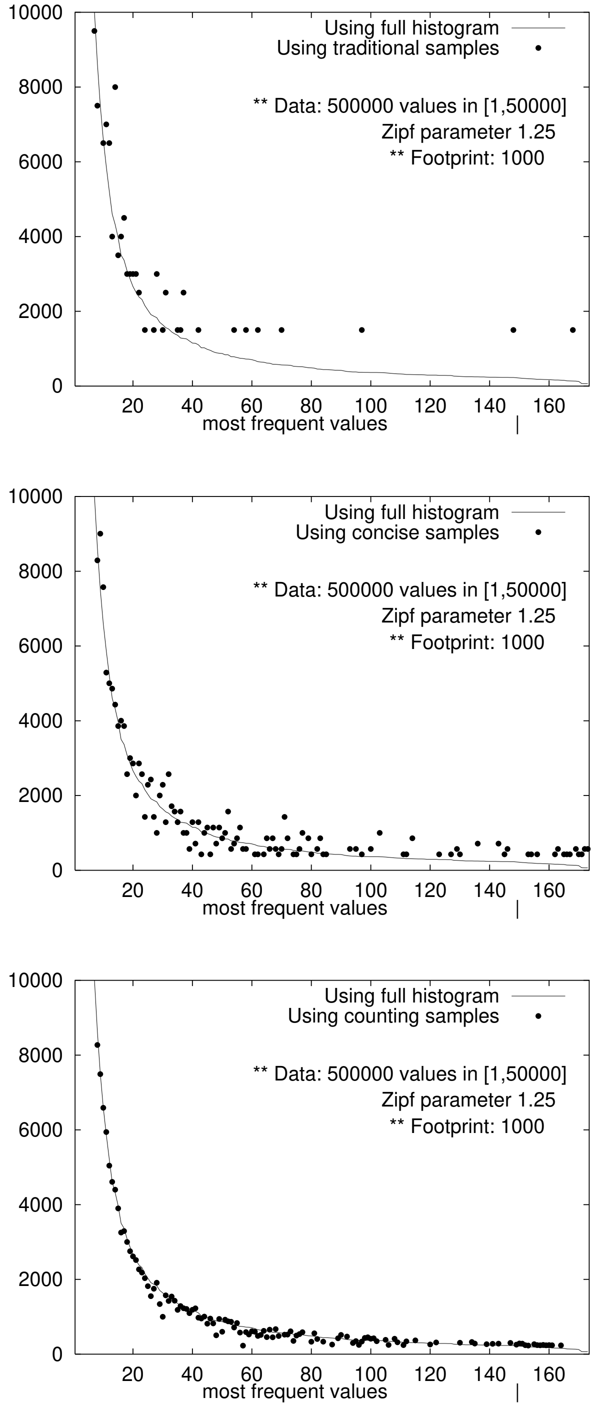


Figure 6: Comparison of traditional, concise, and counting samples on a distribution with zipf parameter 1.25, using footprint 1000.

Table 2: Measured data for the hot-list algorithm experiments in Figures 4–6.

Figure 4	flips	lookups	raises	sample-size	threshold	reported
Using concise samples	0.014	0.008	56	388	1283	18
Using counting samples	0.006	1.000	60	n/a	1881	20
Using traditional samples	0.003	0.000	n/a	100	n/a	9

Figure 5	flips	lookups	raises	sample-size	threshold	reported
Using concise samples	0.040	0.024	40	1813	275	95
Using counting samples	0.053	1.000	47	n/a	541	92
Using traditional samples	0.025	0.000	n/a	1000	n/a	52

Figures 6	flips	lookups	raises	sample-size	threshold	reported
Using concise samples	0.066	0.040	33	3498	140	108
Using counting samples	0.046	1.000	38	n/a	227	122
Using traditional samples	0.025	0.000	n/a	1000	n/a	38

are the number of threshold raises, the final sample-size, the final threshold, and the number of values reported. The number of raises and the final threshold are larger when using counting samples than when using concise samples since the counting sample tends to hold fewer values: its counting of all subsequent occurrences implies that most values in the sample are represented as $\langle$value, count$\rangle$ pairs and not as singletons.

6 Conclusions

Providing an immediate, approximate answer to a query whose exact answer takes many orders of magnitude longer to compute is an attractive option in a number of scenarios. We have presented a framework for an approximate query engine that observes new data as it arrives and maintains small synopses on that data. We have described metrics for evaluating such synopses.

We introduce and study two new sampling-based synopses: concise samples and counting samples. We quantify their advantages in sample-size over traditional samples with the same footprint in the best case, in the general case, and in the case of exponential and zipf distributions. We present an algorithm for the fast incremental maintenance of concise samples regardless of the data distribution, and experimental evidence that the algorithm achieves a sample-size within 1%–28% of that of recomputing the concise sample from scratch at each insert to the data warehouse. The overheads of the maintenance algorithm are shown to be quite small. For counting samples, we present an algorithm for the fast incremental maintenance under both insertions and deletions, with provable guarantees regardless of the data distribution. Random samples are useful in a number of approximate query answers scenarios. The confidence for such an approximate answer increases with the size of the samples, so using concise or counting samples can significantly increase the confidence as compared with using traditional samples.

Finally, we consider the problem of providing fast approximate answers to hot list queries. We present algorithms based on using traditional samples, concise samples, and counting samples. These are the first incremental algorithms for this problem; moreover, we provide analysis and experiments showing their effectiveness and overheads. Using counting samples is shown to be the most accurate, and far superior to using traditional samples; using concise samples falls in between: nearly matching counting samples at high skew but nearly matching traditional samples at very low skew. On the other hand, the overheads are the smallest using traditional samples, and the largest using counting samples. We show both with analysis and experiments that the cost incurred when raising a threshold can be amortized across the entire sequence of data warehouse updates. We believe that using concise samples may offer the best choice when considering both accuracy and overheads.

In this paper, we have assumed a batch-like processing of data warehouse inserts, in which inserts and queries do not intermix (the common case in practice). To address the more general case (which may soon be the more common case), issues of concurrency bottlenecks need to be addressed.

Future work is to explore the effectiveness of using concise samples and counting samples for other concrete approximate answer scenarios. More generally, the area of approximate query answers is in its infancy, and many new techniques are needed to make it an effective alternative option to traditional query answers. In [GPA+98], we present some recent progress towards developing an effective approximate query answering engine.

Acknowledgments

This work was done while the second author was a member of the Information Sciences Research Center, Bell Laboratories, Murray Hill, NJ USA. We thank Vishy Poosala for many discussions related to this work. We also thank S. Muthukrishnan, Rajeev Rastogi, Kyuseok Shim, Jeff Vitter and Andy Witkowski for helpful discussions related to this work.

References

[AMS96] N. Alon, Y. Matias, and M. Szegedi. The space complexity of approximating the frequency moments. In *Proc. 28th ACM Symp. on the Theory of Computing*, pages 20–29, May 1996.

[Ant92] G. Antoshenkov. Random sampling from pseudo-ranked B+ trees. In *Proc. 18th International Conf. on Very Large Data Bases*, pages 375–382, August 1992.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. 20th International Conf. on Very Large Data Bases*, pages 487–499, September 1994.

[AZ96] G. Antoshenkov and M. Ziauddin. Query processing and optimization in Oracle Rdb. *VLDB Journal*, 5(4):229–237, 1996.

[BDF+97] D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. Ioannidis, H. V. Jagadish, T. Johnson, R. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The New Jersey data reduction report. *Bulletin of the Technical Committee on Data Engineering*, 20(4):3–45, 1997.

[BM96] R. J. Bayardo, Jr. and D. P. Miranker. Processing queries for first-few answers. In *Proc. 5th International Conf. on Information and Knowledge Management*, pages 45–52, 1996.

[BMUT97] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 255–264, May 1997.

[FJS97] C. Faloutsos, H. V. Jagadish, and N. D. Sidiropoulos. Recovering information from summary data. In *Proc. 23rd International Conf. on Very Large Data Bases*, pages 36–45, August 1997.

[Fla85] P. Flajolet. Approximate counting: a detailed analysis. *BIT*, 25:113–134, 1985.

[FM83] P. Flajolet and G. N. Martin. Probabilistic counting. In *Proc. 24th IEEE Symp. on Foundations of Computer Science*, pages 76–82, October 1983.

[FM85] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Computer and System Sciences*, 31:182–209, 1985.

[GM95] P. B. Gibbons and Y. Matias, August 1995. Presentation and feedback during a Bell Labs-Teradata presentation to Walmart scientists and executives on proposed improvements to the Teradata DBS.

[GM97] P. B. Gibbons and Y. Matias. Synopsis data structures, concise samples, and mode statistics. Manuscript, July 1997.

[GMP97a] P. B. Gibbons, Y. Matias, and V. Poosala. Aqua project white paper. Technical report, Bell Laboratories, Murray Hill, New Jersey, December 1997.

[GMP97b] 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.

[GPA+98] P. B. Gibbons, V. Poosala, S. Acharya, Y. Bartal, Y. Matias, S. Muthukrishnan, S. Ramaswamy, and T. Suel. AQUA: System and techniques for approximate query answering. Technical report, Bell Laboratories, Murray Hill, New Jersey, February 1998.

[HHW97] 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.

[HK95] M. Hofri and N. Kechris. Probabilistic counting of a large number of events. Manuscript, 1995.

[HNSS95] 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.

[IC93] Y. E. Ioannidis and S. Christodoulakis. Optimal histograms for limiting worst-case error propagation in the size of join results. *ACM Transactions on Database Systems*, 18(4):709–748, 1993.

[Ioa93] Y. E. Ioannidis. Universality of serial histograms. In *Proc. 19th International Conf. on Very Large Data Bases*, pages 256–267, August 1993.

[IP95] Y. E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 233–244, May 1995.

[Mat92] Y. Matias. *Highly Parallel Randomized Algorithmics*. PhD thesis, Tel Aviv University, Israel, 1992.

[Mor78] R. Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21:840–842, 1978.

[MSY96] Y. Matias, S. C. Sahinalp, and N. E. Young. Performance evaluation of approximate priority queues. Presented at *DIMACS Fifth Implementation Challenge: Priority Queues, Dictionaries, and Point Sets*, organized by D. S. Johnson and C. McGeoch, October 1996.

[MVN93] Y. Matias, J. S. Vitter, and W.-C. Ni. Dynamic generation of discrete random variates. In *Proc. 4th ACM-SIAM Symp. on Discrete Algorithms*, pages 361–370, January 1993.

[MVY94] Y. Matias, J. S. Vitter, and N. E. Young. Approximate data structures with applications. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 187–194, January 1994.

[OR89] F. Olken and D. Rotem. Random sampling from B^+ trees. In *Proc. 15th International Conf. on Very Large Data Bases*, pages 269–277, 1989.

[OR92] F. Olken and D. Rotem. Maintenance of materialized views of sampling queries. In *Proc. 8th IEEE International Conf. on Data Engineering*, pages 632–641, February 1992.

[PIHS96] 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.

[Pre97] D. Pregibon. Mega-monitoring: Developing and using telecommunications signatures, October 1997. Invited talk at the *DIMACS Workshop on Massive Data Sets in Telecommunications*.

[Vit85] J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.

[VL93] S. V. Vrbsky and J. W. S. Liu. Approximate—a query processor that produces monotonically improving approximate answers. *IEEE Trans. on Knowledge and Data Engineering*, 5(6):1056–1068, 1993.

[WVZT90] 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.