Tracking Join and Self-Join Sizes in Limited Storage*

Noga Alon
Tel Aviv University
Tel Aviv, Israel
noga@math.tau.ac.il

Phillip B. Gibbons
Intel Research Pittsburgh
Pittsburgh, PA USA
phillip.b.gibbons@intel.com

Yossi Matias
Tel Aviv University
Tel Aviv, Israel
matias@math.tau.ac.il

Mario Szegedy
Rutgers University
Piscataway, NJ USA
szegedy@cs.rutgers.edu

March, 2002

Abstract

This paper presents algorithms for tracking (approximate) join and self-join sizes in limited storage, in the presence of insertions and deletions to the data set(s). Such algorithms detect changes in join and self-join sizes without an expensive recomputation from the base data, and without the large space overhead required to maintain such sizes exactly. Query optimizers rely on fast, high-quality estimates of join sizes in order to select between various join plans, and estimates of self-join sizes are used to indicate the degree of skew in the data.

For self-joins, we consider two approaches proposed in [Alon, Matias, and Szegedy. The Space Complexity of Approximating the Frequency Moments. JCSS, vol. 58, 1999, p.137-147], which we denote *tug-of-war* and *sample-count*. We present fast algorithms for implementing these approaches, and extensions to handle deletions as well as insertions. We also report on the first experimental study of the two approaches, on a range of synthetic and real-world data sets. Our study shows that tug-of-war provides more accurate estimates for a given storage limit than sample-count, which in turn is far more accurate than a standard sampling-based approach. For example, tug-of-war needed only 4–256 memory words, depending on the data set, in order to estimate the self-join size to within a 15% relative error; on average, this is over 4 times (50 times) fewer memory words than needed by sample-count (standard sampling, resp.) to obtain a similar accuracy.

For joins, we propose schemes based on maintaining a small *signature* of each relation independently, such that join sizes can be quickly and accurately estimated between any pair of relations using only these signatures. We show that taking random samples for join signatures can lead to inaccurate estimation unless the sample size is quite large; moreover, we show that no other signature scheme can significantly improve upon sampling without further assumptions. These negative results are shown to hold even in the presence of sanity bounds. On the other hand, we present a fast join signature scheme based on tug-of-war signatures that provides guarantees on join size estimation as a function of the self-join sizes of the joining relations; this scheme can significantly improve upon the sampling scheme.

*To appear in the *Journal of Computer and System Sciences*, special issue of selected papers from PODS'99.

1 Introduction

The degree of *skew* in a data set describes how far the frequency distribution of the items in the data set differ from the uniform distribution. The degree of skew represents important demographic information about the data, and is used to guide the computation in several applications of modern database systems. In a relational database, the size of the self-join on an attribute in a relation R is a well-studied measure of the degree of skew in the attribute values ocurring in R. The self-join size (also called the second frequency moment) on an attribute in R with value domain D is $\sum_{i \in D} f_i^2$, where f_i is the frequency of attribute value i in R. Ioannidis and Poosala [IP95] have advocated using self-join sizes for error estimation in the context of estimating query result sizes and access plan costs. Haas *et al* [HNSS95] advocate its use for selecting between sampling-based algorithms for estimating the number of distinct attribute values in a relation.

Self-join sizes of relations can also be used to bound the join size of any pair of such relations, as follows. Consider the join of relations R_1 and R_2 on a joining attribute with value domain D. For $i \in D$, let f_i and g_i be the frequency of the ith value in R_1 and R_2, respectively. Then the join size, $|R_1 \bowtie R_2| = \sum_{i \in D} f_i \cdot g_i$, satisfies

Fact 1.1

$$|R_1 \bowtie R_2| \leq \frac{\mathrm{SJ}(R_1) + \mathrm{SJ}(R_2)}{2},$$

where $\mathrm{SJ}(R_1) = |R_1 \bowtie R_1|$ *and* $\mathrm{SJ}(R_2) = |R_2 \bowtie R_2|$ *are the self-join sizes on the joining attribute.*

Proof. Note that for any real numbers x and y, $(x-y)^2 \geq 0$. Thus $x^2 - 2xy + y^2 \geq 0$, i.e., $(x^2 + y^2)/2 \geq xy$. Hence $\sum_{i \in D} f_i g_i \leq \sum_{i \in D} (f_i^2 + g_i^2)/2 = (\sum_{i \in D} f_i^2 + \sum_{i \in D} g_i^2)/2 = (\mathrm{SJ}(R_1) + \mathrm{SJ}(R_2))/2$. ■

For many distributions, such as zipfian and exponential, the self-join size uniquely determines the parameter of the distribution.

Fact 1.2 *The self-join size for an exponential distribution uniquely determines the parameter of the distribution.*

Proof. Consider an attribute A in a relation R of size n that is distributed exponentially, i.e., the ith most popular value for attribute A occurs with frequency $n(\alpha-1)\alpha^{-i}$. Then $\mathrm{SJ}(R) = \sum_i (n(\alpha - 1)\alpha^{-i})^2 = n^2(\alpha-1)^2 \sum_i (\alpha^2)^{-i} = n^2(\alpha-1)^2/(\alpha^2-1) = n^2(\alpha-1)/(\alpha+1)$. It follows that $\alpha = (n^2 + \mathrm{SJ}(R))/(n^2 - \mathrm{SJ}(R))$. ■

In the statistics literature, the self-join size is referred to as the *repeat rate* or *Gini's index of homogeneity* needed in order to compute the *surprise index* of the sequence (see, e.g., [Goo89]).

The self-join size can be computed in one pass over the data by computing a full histogram of the data, and then summing the squares of the frequency counts for each attribute value. However, this requires storage proportional to the number of distinct attribute values, which may be prohibitively large. Hence, we seek alternative approaches.

Tracking self-join sizes. In this paper, we first study algorithms for tracking (approximate) self-join sizes in limited storage in the presence of insertions and deletions to the database. Alon, Matias, and Szegedy [AMS99] proposed two approaches for tracking self-join sizes in the presence of insertions, which we denote as *sample-count* and *tug-of-war*, and presented asymptotic upper bounds on the space $s(\epsilon, \delta)$[1] required to guarantee accuracy within ϵ relative error with confidence probability $1 - \delta$ (for $\epsilon > 0$ and $0 < \delta < 1$). For example, they showed that sample-count requires $\Theta(\sqrt{t})$ space to guarantee a constant factor relative error with confidence $> \frac{1}{2}$ for any distribution with t distinct values, whereas tug-of-war requires only $O(1)$ memory words. The focus in [AMS99] was entirely on space bounds and accuracy; time was not explicitly considered.

In this paper, we extend the results in [AMS99] in three ways. First, we show how to handle deletions as well as insertions. Specifically, the input to our tracking algorithms is a sequence of operations, where each operation is either an insertion of a new data item or a deletion of an existing data item. Second, we show how the sample-count approach can be implemented in constant amortized time (with high probability) per operation, independent of s. (The tug-of-war approach takes $O(s)$ time per operation.) Third, we present the first experimental study of the two approaches, comparing them with a standard sampling-based approach, for a dozen real-world and well-studied synthetic data sets. Our study demonstrates the practical utility of the sample-count and tug-of-war algorithms, by showing that good estimates are obtained while using only a small fraction of the memory required to maintain the exact self-join size. The study reveals how the algorithms perform on common distributions, something not revealed by the *worst case* analysis provided in [AMS99]. For example, we show that unless the self-join size is predominantly determined by very few items, the standard sampling approach is far less accurate than the other two approaches. Moreover, our experiments indicate that tug-of-war is more accurate than sample-count on a wide variety of data sets, although the accuracy of sample-count is often close and sometimes better than that of tug-of-war. The relative closeness of sample-count and tug-of-war contrasts with the large gap in the asymptotic bounds indicated above. In order to verify that this worst case gap can occur (and is not simply an artifact of the upper bound analysis), we construct a pathological data set for which sample-count converges particularly slowly, in contrast to tug-of-war. Finally, our results show the amount of memory needed to obtain a given accuracy on common distributions. For example, tug-of-war needed only 4–256 memory words, depending on the data set, in order to estimate the self-join size to within a 15% relative error, and additional memory words lead quickly to more accurate estimates.

Tracking join sizes. Next, we study algorithms for tracking (approximate) join sizes in limited storage. Query optimizers rely on fast, highly-accurate join size estimates in order to select between various join plans. To avoid the quadratic blow-up inherent in maintaining separate data structures for each possible pair of joining relations, we instead consider schemes based on tracking each relation independently. The goal is to maintain a small signature of each relation such that join sizes can be quickly and accurately estimated between any pair of relations using only these signatures.[2] We show that taking random samples for join signatures can lead to inaccurate estimation unless the sample size is quite large. Moreover, we prove a (nontrivial) lower bound that shows that no other signature scheme can provide significantly better estimation guarantees without further assumptions. These

[1] Although Alon et al. presented space bounds in terms of the number of bits, we will present space bounds in terms of the number of $\Theta(\log n)$-bit memory words, unless noted otherwise.

[2] In this paper, we restrict our attention to equality joins between pairs of relations, where each join of interest is on the same attribute A. In general, if there are equality joins on an attribute A in a relation R, and other equality joins on an attribute B in R, then separate signatures would be needed for A and B in the scheme we propose.

negative results are shown to hold even in the presence of a sanity bound B.[3] Specifically, for any B such that $n \leq B \leq \frac{n^2}{2}$, where n is the size of each relation, we show that $\Omega(\frac{n^2}{B})$ bits are required. Sanity bounds are popular for estimation algorithms, but tend to be ignored when proving lower bounds. On the other hand, we present a join signature scheme based on tug-of-war (self-join) signatures that provides guarantees on join size estimation as a function of the self-join sizes of the joining relations; this scheme can significantly improve upon the sampling scheme whenever the self-join sizes are smaller than $n\sqrt{B}$.

The performance and accuracy bounds of the algorithms in this paper are valid for any data distributions.

Related work. Tracking algorithms and other general data reduction techniques have a long history; see [BDF+97] for a recent survey. [GM99] presented a formal framework for evaluating such sublinear space "synopsis" data structures, and a survey of some of the results in this area. There has been a flurry of recent work in approximate query answering (e.g., [VL93, Olk93, BDF+97, HHW97, GM98, AGPR99, HH99, VW99, IP99, AGP00, GLR00, CCMN00, CGRS00, MVW00, CDN01, LM01, Gib01, GKS01]). The work in [HHW97, AGPR99, HH99, IP99, CGRS00] looked at the problem of providing approximate answers to queries seeking aggregates (e.g., `count`, `sum`, `avg`) of attribute values for the tuples satisfying a predicate that occur in the join of multiple relations. The `count` aggregate (over joins but with no other predicates) corresponds to the join size estimation problem considered in this paper. However, approaches based on histograms [IP99] or wavelets [CGRS00] do not provide any good accuracy guarantees. Online sampling approaches [HHW97, HH99] do not perform any tracking, and instead incur large overheads for sampling at estimation time. Finally, previous precomputed sampling-based approaches are accurate only for foreign-key joins [AGPR99], and otherwise require large space for accurate estimation for arbitrary equality joins.

There is an extensive literature on join size estimation (e.g., [HÖT88, LNS90, HNSS93, LN95, GGMS96]). As in the online sampling approaches discussed above, the techniques presented in these papers target the traditional approach of estimating the join sizes without the benefit of precomputed signatures, and hence also incur large overheads at estimation time. For example, sampling-based approaches take samples of the databases at the time of estimation; such sampling is slow due to the random disk accesses involved. In contrast, our tracking approaches do not incur disk accesses at estimation time. Also, they adapt incrementally to database updates, in contrast to previous approaches that recompute from scratch at each estimation time. (Some of our analysis holds for this traditional scenario as well.) Poosala [Poo97] proposed join size estimation using signatures that are the Compressed histogram of each relation. (Such histograms can be maintained incrementally using the algorithm in [GMP97].) However, there are no good guarantees on the accuracy of such estimations.

The sample-count approach is somewhat reminiscent of the algorithm in [GM98] for maintaining "counting samples". However, counting samples are used to track the top-k most popular values in a data set, and not the self-join size. They permit a value to be selected for the sample at most once, whereas it is crucial for the accuracy of sample-count's self-join size estimation that the same value

[3] *Sanity bounds* stipulate a lower bound on the quantity being estimated, such that estimation errors are analyzed only for quantities above this lower bound (see, e.g., [LN95, LNS90, GGMS96]), presumably the range of interest to the application making use of the estimate. Since estimating small quantities is often considerably more difficult than estimating large quantities, the use of sanity bounds may improve the estimation guarantees considerably.

can be selected for the sample many times. The top-k list attempts to report the top k values and their frequency, whereas the self-join size reports a single estimator. This allows the latter to apply averaging and median techniques over multiple intermediate estimators, all within the limited storage.

Outline. The rest of the paper is organized as follows. In Section 2 we consider the sample-count and tug-of-war approaches. For each approach, we first describe the algorithm as presented in [AMS99], and then present our contributions for handling deletions and obtaining good time bounds. We also present a new lower bound for a standard sampling-based algorithm. Section 3 presents our experimental study of the three algorithms for self-join size estimation. Section 4 presents our new signature scheme for join size estimation, along with our upper and lower bound analyses. Finally, concluding remarks appear in Section 5.

2 Tracking Self-Join Sizes

In this section we present three algorithms for approximating self-join sizes in limited storage: sample-count (Section 2.1), tug-of-war (Section 2.2), and naive-sampling (Section 2.3). Let $R = (v_1, v_2, \ldots, v_n)$ be a sequence of n values on which we are to estimate the self-join size, where each v_i is a member of $D = \{1, 2, \ldots, t\}$. The basic idea in both sample-count and tug-of-war is a natural one. In order to estimate the self-join size, $\mathrm{SJ}(R)$, a random variable is defined that can be computed under a given space constraint, whose expected value is $\mathrm{SJ}(R)$, and whose variance is relatively small. The desired result is then obtained by considering sufficiently many such random variables, partitioning them into groups, computing the average within each group, and then taking the median of the group averages. In contrast, naive-sampling selects a random sample of the items in R, computes the self-join size for the sample, and then scales up the result, so that the expected value of the estimate is $\mathrm{SJ}(R)$.

To handle the general tracking scenario, we consider a sequence of operations on a multiset R, initially empty, where each operation is either

- insert(v): insert a value $v \in D$ into R,
- delete(v): delete an occurrence of the value $v \in D$ from R, or
- query: compute an estimate of the self-join size of R.

2.1 Algorithm sample-count

Alon et al. [AMS99] presented the following approach to estimating the self-join size of a sequence R (insertions only).

1. For $i \in \{1, 2, \ldots, s_1\}$ and $j \in \{1, 2, \ldots, s_2\}$, compute an independent random variable $X_{i,j}$ as follows:
   - Choose a random member v_p of the sequence R, where the index p is chosen randomly and uniformly among the numbers $1, 2, \ldots, n$; suppose that $v_p = l$ ($\in D$).
   - Let $r_{i,j} = |\{q : q \geq p, v_q = l\}|$ (≥ 1) be the number of occurrences of l among the members of the sequence R following v_p (inclusive).

- Let $X_{i,j} = n(2r_{i,j} - 1)$.

2. For $j \in \{1, \ldots, s_2\}$, let Y_j be the average of $\{X_{1,j}, X_{2,j}, \ldots, X_{s_1,j}\}$.

3. Let the estimate Y be the median of $\{Y_1, \ldots, Y_{s_2}\}$.

The algorithm has two parameters: s_1 determines the accuracy of the result, and s_2 determines the confidence. Let $s = s_1 \cdot s_2$. The algorithm uses $\Theta(s)$ memory words. The above description assumes that n is known in advance. If n is not known, [AMS99] proposed that after each insertion, each sample point is replaced by the next point independently with probability $\frac{1}{n+1}$, where n is the current length of the sequence.

Note that each time a value v is inserted that occurs k times among the s selected sample points, k different $r_{i,j}$ are incremented. Thus a straightforward implementation of this algorithm using counters requires $\Omega(k)$ time to process the insertion. Large k will be expected for highly-skewed data. In the worst case, all members are the same type, resulting in $\Omega(s)$ time to process each insertion. Moreover, the straightforward implementation of the process above to handle unknown n requires $\Theta(s)$ time per insertion. Finally, important algorithmic details are missing, and the approach does not specify how to handle deletions.

In the remainder of this section, we show how the above deficiencies can be remedied, by presenting an algorithm for implementing this approach that both handles deletions and achieves $O(1)$ amortized time per update with high probability.

New results. New update operations (i.e., insertions and deletions) are expected to occur far more frequently than new queries. Thus our goal is to minimize update times, while keeping reasonable query times and preserving the high quality of the estimates. We achieve this goal with our improved sample-count algorithm, depicted in Figure 1. Steps 1–5 perform initialization, including selecting the initial s random positions. Then the main loop, starting with step 6, processes the operations. Insert operations are handled by steps 7–19. Delete operations are handled by steps 20–26. Query operations are handled by steps 27–32.

Various data structures are used to permit fast access to certain properties of the s sample points. For $i = 1, \ldots, s$, Pos[i] holds the random position selected (step 5). This means that the i'th sample point will have the value v of the Pos[i]'th insert. We say that i *entered* the sample when that insert is processed, and that subsequently it is *in* the sample. (As discussed below for deletions, i may be later removed from the sample.) We say that v *occurs* in the sample; v is stored in Val[i] (step 17). If v occurs in the sample, then S_v is the set of all i in the sample that have value v. Otherwise, S_v is undefined. If m has been selected as a sample point position, but fewer than m insertions have been processed thus far, then P_m is the set of all i that selected position m. (We do not expect many duplicate selections of the same m, so typically P_m contains only a single i.) Otherwise, P_m is undefined. The defined P_m are stored in a look-up table of size $\Theta(s)$, using m as a look-up key.

To avoid the problem described above of having to increment up to s of the r-counters with each insert, we use the following approach. For each value v occurring in the sample, we maintain a running count N_v of the number of occurrences of v (steps 19 and 23). We store these N_v in a look-up table of size $\Theta(s)$, using v as the look-up key. For each i in the sample, EntryNv[i] holds the value of N_v just prior to when i entered the sample (step 17). Then in response to a query, we can compute the r-counter for sample point i by subtracting EntryNv[i] from the current N_v (step 30). Note that

Algorithm sample-count

inputs: Sequence of insert, delete, and query operations. Parameters s_1 and s_2. Let $s = s_1 \cdot s_2$.

```
n := 0;   m := 0        //size of R and number of inserts
initialize empty look-up tables for N_v (on key v), S_v (on key v), and P_m (on key m)
for all i ∈ {1, 2, ..., s} {
   select uniformly at random a position p from {1, 2, ..., s log s}
   Val[i] := ⊥;   Pos[i] := p;   add i to P_p
}
for each operation in the sequence {
   if the operation is an insert(v) then {
      n := n + 1;   m := m + 1
      if P_m is defined then {        // if this position has been selected
         for all i ∈ P_m {
            select a random replacement position p        // reservoir sampling: see text
            Pos[i] := p;   add i to P_p
            if Val[i] ≠ ⊥ then {        // if replacing an existing sample point
               remove i from S_Val[i]
               if S_Val[i] is now empty then undefine S_Val[i] and N_Val[i]
            }
         }
         if S_v is not defined then define S_v := empty and N_v := 0
         for all i ∈ P_m { Val[i] := v;   EntryNv[i] := N_v;   add i to head of S_v }
         undefine P_m
      }
      if S_v is defined then N_v := N_v + 1        // if v occurs in the sample
   }
   else if the operation is a delete(v) then {
      n := n − 1
      if S_v is defined then {        // if v occurs in the sample
         N_v := N_v − 1
         let S* be the (possibly empty) subset of S_v with EntryNv[i] = N_v
         for all i ∈ S* {        // a sample point is being deleted
            Val[i] := ⊥;   remove i from S_v;   if S_v is now empty then undefine S_v and N_v
         }
      }
   }
   else if the operation is a query then {
      j := 1;   Sum := 0;   Num := 0
      for i ∈ {1, ..., s} {        // compute the averages
         if Val[i] ≠ ⊥ then { Sum := Sum + N_Val[i] − EntryNv[i];   Num := Num + 1 }
         if i mod s_1 = 0 then { Y_j := n(2 Sum/Num − 1);   j := j + 1;   Sum := 0;   Num := 0 }
      }
      Let the estimate Y be the median of {Y_1, ..., Y_s2}
   }
}
```

Figure 1: The improved sample-count algorithm

because we have only $O(s)$ space, we maintain N_v's only for v's occurring in the current sample. Thus N_v will start accumulating only when v first occurs in the sample, and although this first sample point may later be deleted, N_v will continue to accumulate as long as v occurs in the sample.

With the above descriptions, the steps for insertions (ignoring for now steps 10–15) are straightforward. Likewise, the steps for queries are straightforward. Note that, according to the Alon et al. approach, the goal is to compute the median of the set of Y_j's, where each $Y_j = \frac{1}{s_1}\sum_{i=1}^{s_1} X_{i,j} = n(\frac{2}{s_1}\sum_{i=1}^{s_1} r_{i,j} - 1)$. Steps 28–31 compute these Y_j's, where the only twist is that we ignore i that are not in the sample. Such i can occur when we have not yet encountered the Pos[i]'th insert, or when the sample point has been discarded due to a delete operation. Then step 32 computes the desired median.

We now describe how we handle deletions. We will assume that the adversary cannot adapt the sequence in response to the random choices made by our algorithm. Note that we have defined our tracking problem to be one of maintaining a multiset R under insertions and deletions, and producing estimates of $\mathrm{SJ}(R)$. This formulation is based on the observation that each sequence member can be replaced by its value, for the purposes of estimating the self-join. Thus for any delete(v) operation, we can assume without loss of generality that the member to be deleted is the one with value v that was the last one to be inserted (and not yet deleted). Using this assumption, we can represent each sequence of insertions and deletions by a *canonical sequence* which consists of insertions only, but possibly contains nil values. Let $\hat{A}$ be a (prefix) sequence consisting of insertions and deletions. We obtain its canonical sequence A' by scanning $\hat{A}$ from left to right; whenever we see delete(v), we replace it with a nil value, and in addition we find the nearest member to the left of it with value v and replace it with a nil value as well. The non-nil values in A' constitute the multi-set of values that remain in the relation after processing the sequence $\hat{A}$. Let A be the subsequence of A' when the locations with the nil values are ignored. In this way, we have reduced the scenario with insertions and deletions to one with insertions only.

Our task then is to have our algorithm process the sequence $\hat{A}$, containing both insertions and deletions, so that the end result is as if the input had been the insertion-only sequence A. In other words, the deletion should reverse the most recent undeleted insertion of the same value. Accordingly, first we decrement the running count, n, of the size of R (step 21). If v occurs in the sample, we decrement N_v (step 23). If the insert(v) to be set to nil is a sample point, then remove it from the sample. In order to detect this scenario quickly, we maintain S_v as a doubly-linked list, ordered from the most recent (undeleted) i with value v to enter the sample to the least recent. The heads of these lists are kept in a look-up table of size $\Theta(s)$, using v as the look-up key. Any i in S_v with $\mathrm{EntryNv}[i] = N_v$ is to be removed (steps 24–26). In this way, we succeed in reversing the most recent undeleted insert(v).

The end result is as if we have processed A, except that certain sample points were dropped and may not have been replaced. As long as the number of delete operations in any prefix of a sequence $\hat{A}$ is at most $1/5$ of the length of $\hat{A}$, then Chernoff bounds can be used to show that with high probability the number of remaining sample points after processing the sequence $\hat{A}$ is at least $s/2$. As a result, we obtain accuracy that is provably close to that obtained for insertions only, in which the number of sample items is guaranteed to be s. Thus, we obtain basically the same estimation quality guarantees as in [AMS99], as described in the following theorem.

Theorem 2.1 *Consider* **Algorithm sample-count** *run with parameters s_1 and s_2 on a sequence σ*

of at least $s \log s$ insertions ($s = s_1 \cdot s_2$) ending in a query, where the number of insertions in σ exceeds by at least a factor of 4 the number of deletions in σ. Let R be the multiset resulting after σ. Then the estimate Y computed by the algorithm satisfies:

$$\mathrm{Prob}\left(\frac{|Y - \mathrm{SJ}(R)|}{\mathrm{SJ}(R)} \le \frac{4t^{1/4}}{\sqrt{s_1}}\right) \ge 1 - 2^{-s_2/2},$$

where t is the size of the value domain. Moreover, **Algorithm sample-count** *can be implemented such that insert and delete operations are processed in $O(1)$ amortized time per operation with high probability, and each query is answered in $O(s)$ time. The algorithm uses $O(s)$ memory words.*

Given their similarity to the estimation guarantees and analysis in [AMS99], we omit the proof details for the estimation guarantees of Algorithm sample-count, and focus on the time bounds. In particular, we now describe the steps to deal with an ever-increasing n. Recall that we need to maintain the invariant that each i selects a position at random from $\{1, \ldots, n\}$. After each new insertion, we would like to replace each sample point by the next point independently with probability $\frac{1}{n+1}$, without incurring the $\Theta(s)$ time per insert. For this, we use the following *skipping* approach employed in reservoir sampling [Vit85]. Considering each i as its own reservoir sample of size 1, the skipping approach computes the next random position that would succeed in replacing the current point. That is, if $\mathrm{Pos}[i] = n$, then this position is replaced by position $n+1$ with probability $\frac{1}{n+1}$, by position $n+2$ with probability $(1 - \frac{1}{n+1})\frac{1}{n+2}$, etc. The skipping approach performs a binary search to find the appropriate position, taking $O(\log n)$ time with high probability, but then the position selected is expected to be good for the next n insertions. Thus given that $n \ge s \log s$, all s instances of reservoir sampling combined take amortized $O(1)$ time with high probability.

We now discuss the reservoir sampling steps in more detail. $\mathrm{Pos}[i]$ holds the selected future position for i. Initially, this is chosen at random from $\{1, \ldots, s \log s\}$. When the $\mathrm{Pos}[i]$'th insert is processed, it is time to select the new $\mathrm{Pos}[i]$ (steps 11 and 12). This initial application of skipping considers only positions greater than $s \log s$; subsequent applications consider all positions greater than the current $\mathrm{Pos}[i]$. Each time a new $\mathrm{Pos}[i]$ becomes the current insert, we need to discard the existing sample point (steps 13–15) before we can add the new sample point (step 17). Note that unlike deletions resulting from the delete operation, the i to be discarded may occur anywhere in $S_{\mathrm{Val}[i]}$. This is why we need a doubly-linked list for the S_v's. Also note that because we are not actually deleting a value from R, only from the sample, we do not need to modify $N_{\mathrm{Val}[i]}$. To wrap up this time analysis, we observe that the look-up tables can be implemented as standard hash tables with chaining, resulting in our claimed $O(1)$ amortized time bound with high probability for processing updates.

In order to achieve the $O(s)$ time per query stated in the theorem, we need to retrieve $N_{\mathrm{Val}[i]}$ in step 30 without using a table look-up. This can be achieved by representing the list S_v using "next" and "prev" arrays, and storing the N_v in an array of size s, indexed by the i at the head of S_v. We walk through this latter array. Each time a valid N_v is encountered, we chase the "next" pointers, broadcasting N_v to all i in S_v, so that each i can learn the corresponding $N_{\mathrm{Val}[i]}$.

Finally, we note that as an alternative to the algorithm and bounds above, we can devise an algorithm with faster query times, at the cost of slower update times, as follows. The basic idea is to maintain each Y_j during updates, so that queries can be done in $O(s_2)$ time. We maintain $k_{v,j}$, the number of i in the sample with value v that are relevant to the sum Y_j. We store $k_{v,j}$ in a look-up table using key v, where the j's for value v are stored as a list at most s_2 long. On an insert(v), the mth insert, we increment $k_{v,\lceil\frac{i}{s_1}\rceil}$ for all $i \in S_m$. Then for all j, add $k_{v,j}$ to Y_j. On a delete(v), reverse these steps. When we replace i's sample point during the reservoir sampling, we decrement $k_{\mathrm{Val}[i],\lceil\frac{i}{s_1}\rceil}$ and subtract $N_{\mathrm{Val}[i]}-$ EntryNv[i] from $Y_{\lceil\frac{i}{s_1}\rceil}$. We also maintain Num$_j$, the number of i in the sample that are relevant to Y_j. Note that $\mathrm{Num}_j = \sum_v k_{v,j}$. On a query, we divide each Y_j by Num$_j$, determine the median Y^*, and let the estimate $Y = n(2Y^* - 1)$. This results in $O(s_2)$ amortized time with high probability for updates, $O(s_2)$ time for queries, $O(s)$ memory words, and the same estimation guarantees as before.

Note that Algorithm sample-count does not require a priori knowledge about the length of the sequence, the size of R at query times, or the number of distinct values in R.

2.2 Algorithm tug-of-war

Alon et al. [AMS99] presented the following alternative approach to estimating the self-join size of a sequence R (insertions only).

1. For $i \in \{1, 2, \ldots, s_1\}$ and $j \in \{1, 2, \ldots, s_2\}$, compute an independent random variable $X_{i,j}$ as follows:
   - Select at random a 4-wise independent mapping $v \mapsto \epsilon_v$, where $v \in \{1, 2, \ldots, t\}$ and $\epsilon_v \in \{-1, 1\}$.
   - Let $Z_{i,j} = \sum_{v=1}^{t} \epsilon_v m_v$, where m_v is the number of members with value v.
   - Let $X_{i,j} = Z_{i,j}^2$.
2. For $j \in \{1, \ldots, s_2\}$, let Y_j be the average of $\{X_{1,j}, X_{2,j}, \ldots, X_{s_1,j}\}$.
3. Let the estimate Y be the median of $\{Y_1, \ldots, Y_{s_2}\}$.

As in sample-count, the algorithm has two parameters: s_1 determines the accuracy of the result, and s_2 determines the confidence. Let $s = s_1 \cdot s_2$. The algorithm uses $\Theta(s)$ memory words.

We denote this the *tug-of-war* approach because each member of the sequence with a value mapping to $+1$ can be viewed as pulling the rope one direction, while each member with a value mapping to -1 can be viewed as pulling the other direction. Note that all members with the same value will be on the same side of the rope. However, the particular random mapping determines which side of the rope they are on. Alon et al. showed that the expected square of the difference in the sizes of the two sides is exactly SJ(A), and that the variance is reasonably small.

A tracking algorithm. The above approach is readily adapted to handle a general sequence of insertions, deletions, and queries. We initialize each $Z_{i,j}$ to 0. We select s hash functions $h_{i,j}$ mapping from $\{1, \ldots, t\}$ to $\{-1, 1\}$. On an insert(v), we add $h_{i,j}(v)$ to $Z_{i,j}$ for all i and j. On a delete(v), we subtract $h_{i,j}(v)$ from $Z_{i,j}$ for all i and j. On a query, we compute Y as above. Denote this algorithm as Algorithm tug-of-war. Assuming that the $h_{i,j}(v)$ can be computed in constant time, we have:

Theorem 2.2 *Consider* **Algorithm tug-of-war** *run with parameters s_1 and s_2 on a sequence σ ending in a query. Let R be the multiset resulting after σ. Let $s = s_1 \cdot s_2$. Then the estimate Y*

computed by the algorithm satisfies:

$$\text{Prob}\left(\frac{|Y - \text{SJ}(R)|}{\text{SJ}(R)} \leq \frac{4}{\sqrt{s_1}}\right) \geq 1 - 2^{-s_2/2}.$$

Moreover, insert, delete, and query operations each take $O(s)$ time, and the algorithm uses $O(s)$ memory words.

The estimation guarantees follow directly from the analysis in [AMS99]. The time and space bounds are immediate. Note that this algorithm does not require a priori knowledge about the length of the sequence, the size of R at query times, or the number of distinct values in R.

2.3 Algorithm naive-sampling

We contrast Algorithm sample-count and Algorithm tug-of-war with the following standard sampling approach (not considered in [AMS99]), denoted below as Algorithm naive-sampling. We consider the simple scenario of a sequence A with only insertions, where the number of insertions, n, is known. The algorithm samples s elements (without replacement) from the sequence, and computes the self-join size, $\text{SJ}(S)$, of the sample set S, by first computing a simple histogram of at most s buckets on the values that occur in the sample set, and then summing the squares of the bucket counts. We then scale $\text{SJ}(S)$ into an estimator X whose expected value is $\text{SJ}(A)$:

$$X = n + \frac{(\text{SJ}(S) - s)n(n-1)}{s(s-1)}.$$

We have the following lower bound on the sample size required to provide a good quality estimate of the self-join size, even in this simplified scenario.

Lemma 2.3 Algorithm naive-sampling *requires a sample of size $\Omega(\sqrt{n})$ to estimate the self-join size to within less than a factor of 2 with high probability.*

Proof. Let R_1 contain n items of different values. Let R_2 contain $n/2$ pairs of items such that each pair contains items with the same value. Members of different pairs have different values. A random sample of R_1 will contain all distinct values, and hence the estimator of the self-join size for R_1 will be n. A random sample of R_2 of size $o(\sqrt{n})$ will, with a sizeable probability p, also contain all distinct values; the estimator for R_2 will also be n on such a sample. On the other hand, $\text{SJ}(R_2) = 2 \cdot \text{SJ}(R_1) = 2n$, so the estimator will be a factor of 2 off with probability at least p. ∎

2.4 Analytical comparison of the algorithms

In both Algorithm sample-count and Algorithm tug-of-war, a single random variable is expected to provide the right estimate. However, in order to guarantee from Theorem 2.1 that for any input set, Algorithm sample-count produces an accurate estimate with high probability, we need to have a sample of size $\Theta(\sqrt{t})$. This improves upon the $\Omega(\sqrt{n})$ lower bound for Algorithm naive-sampling, when $t \ll n$. On the other hand, we see from Theorem 2.2 that only $O(1)$ tug-of-wars suffice for an accurate estimate with high probability. Thus in theory, Algorithm sample-count is inferior to Algorithm tug-of-war in both its space requirement and its simplicity of implementation. However, recall that

Table 1: Data sets and their characteristics

data set	length	domain size	self-join size	type	figure
zipf1.0	$500,000$	$9,994$	$4.30e+09$	statistical	2
zipf1.5	$120,000$	$2,184$	$2.59e+09$	statistical	3,15
uniform	$1,000,000$	$32,768$	$3.15e+07$	statistical	4
mf2	$19,998$	$1,693$	$3.98e+06$	statistical	5
mf3	$19,968$	$2,881$	$6.19e+05$	statistical	6
selfsimilar	$120,000$	200	$3.41e+09$	statistical	7
poisson	$120,000$	39	$9.12e+08$	statistical	8
wuther	$120,952$	$10,546$	$1.12e+08$	text	9
genesis	$43,119$	$2,674$	$2.31e+07$	text	10
brown2	$855,043$	$46,153$	$5.84e+09$	text	11
xout1	$142,732$	$12,113$	$9.17e+07$	geometric	12
yout1	$142,732$	$12,140$	$9.46e+07$	geometric	13
path	$40,800$	$40,001$	$6.80e+05$	artificial	14

Algorithm tug-of-war is somewhat more demanding in its update time, which is proportional to the sample size. More importantly perhaps, the estimation guarantees in Theorems 2.1 and 2.2 apply in general to any input. This leaves open the question as to which of the methods would demonstrate better performance in actual use. The experimental study in the next section provides the first step towards answering this question.

3 An Experimental Study

In this section, we present the results of our experimental study of the three algorithms described in Section 2. We implemented the sample-count, tug-of-war and naive-sampling algorithms, and tested their accuracy on various real-world and synthetic data sequences.

Data sets. Table 1 presents the data sets we studied, and summarizes their properties. The first seven data sets are synthetic data sets comprised of data values drawn from different statistical distributions. The first two are from a Zipfian distribution with parameters 1.0 and 1.5, where the larger parameter implies more skew. The next is from a uniform distribution, followed by two from multifractal distributions, with larger and smaller skew. The last two statistical data sets are from a self-similar distribution and from a Poisson distribution. The next five data sets in the table are real-world data sets. We study three that are text excerpts from well-known literary works such as *Wuthering Heights* and *Genesis*, and two that are coordinates taken from a spatial data set. Finally, we have an artificial data set designed to favor tug-of-war over sample-count.

For each data set, Table 1 lists its length (n), its domain size (t), the actual self-join size, its type (as indicated above), and which figure depicts results for this data set. Note that these data sets span a factor of 50 in lengths, three orders of magnitude in domain sizes, and nearly four orders of magnitude in self-join sizes.

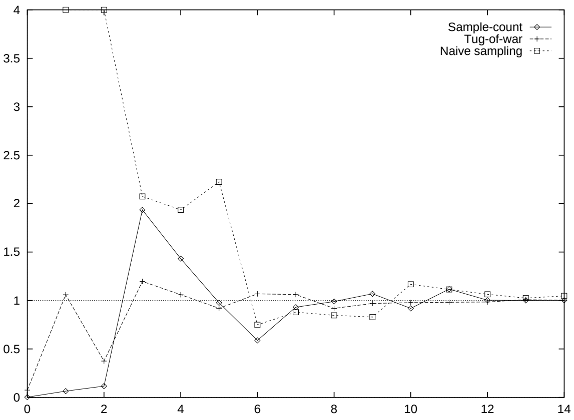

Figure 2: Zipf(1.0) distribution

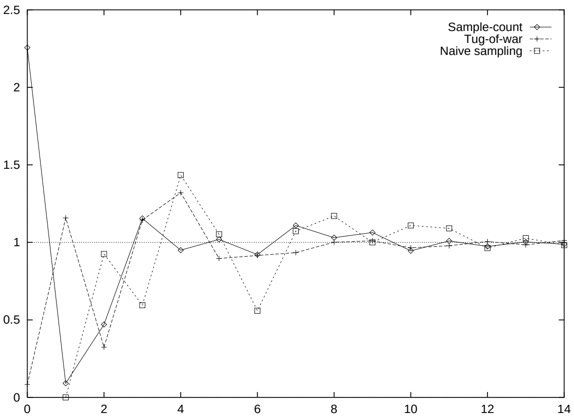

Figure 3: Zipf(1.5) distribution

Plots. The accuracy for the three algorithms was measured for sample sizes ranging from 1 to 16,384, by powers of 2. The results are shown in Figures 2–14. In each plot, the x-axis is labeled with the base two logarithm of the sample size. The y-axis is labeled with the ratio of the estimated size to the actual size of the self-join, i.e., the estimate normalized by the actual. The actual join size is shown as a horizontal line at $y = 1$. For each sample size, we plot the normalized estimate produced by Algorithm sample-count, Algorithm tug-of-war, and Algorithm naive-sampling. For all three algorithms, by the law of large numbers, the normalized estimate must tend to 1 as the sample size grows, since the expectation of each estimator equals the self-join size. Each plotted point corresponds to one run of an algorithm; this seemed appropriate because each estimator is already based on the aggregation of many independent experiments.

3.1 Summary of the results

Figure 2 depicts a common case. Algorithm tug-of-war converges to the actual self-join size at a faster rate than Algorithm sample-count, which converges at a faster rate than Algorithm naive-sampling. As a simple means of quantifying convergence towards a reasonable approximation, we will consider the metric of the minimum sample size each algorithm needed to be within 15% relative error for this and all larger sample sizes. For the Zipf(1.0) data set, tug-of-war needed a sample size of only $2^4 = 16$, sample-count needed $2^7 = 128$, but naive-sampling needed $2^{11} = 2048$.

On the other hand, when we consider a more skewed Zipfian data set (Figure 3), both sample-count and naive-sampling improve, whereas tug-of-war does not improve. The result is that sample-count is roughly comparable to tug-of-war, and both are far better than naive-sampling. By our metric, sample-count needed sample size 16, tug-of-war needed 32, and naive-sampling needed 512.

Interestingly, when we consider a uniform distribution (Figure 4), which has no inherent skew, sample-count does much better than tug-of-war, which is better than naive-sampling, sample size 16 vs. 256 vs. 2048, respectively, for our metric. In both the no-skew and the high-skew cases, sample-count does well because both types of distributions are well-represented by the counts for a few randomly selected positions.

In comparing the higher skew and lower skew multi-fractal data sets (Figures 5 and 6, respectively), we see that both tug-of-war and sample-count are comparable, but naive-sampling does considerably

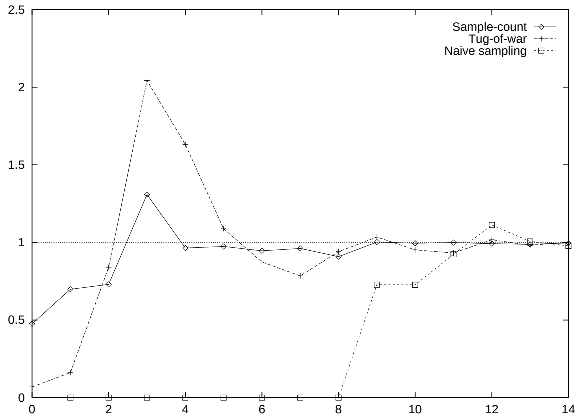

Figure 4: Uniform distribution

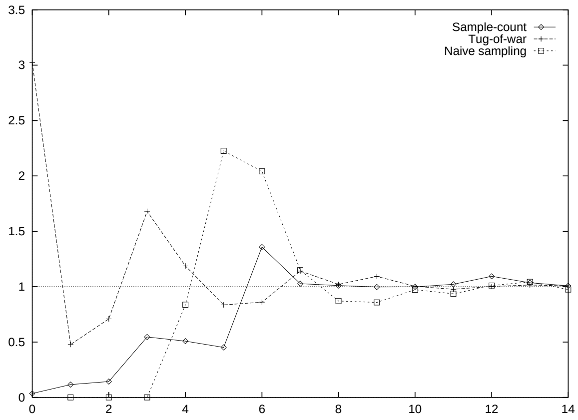

Figure 5: Multi-fractal(20000, 0.2, 12) distribution

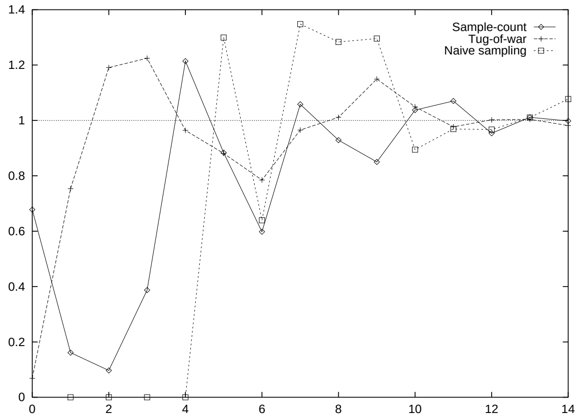

Figure 6: Multi-fractal(20000, 0.3, 12) distribution

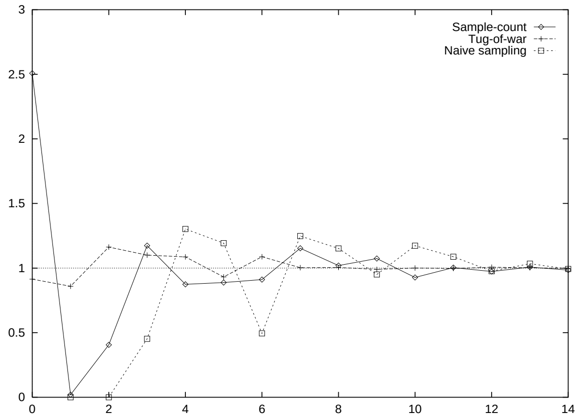

Figure 7: Self-similar distribution

worse on the lower skew data set. In fact, it has yet to converge within 15% even with a sample size of $16,384$, which is over 80% of the size of the entire data set.

The next two data sets compare the algorithms on very small domain sizes. Algorithm naive-sampling is far worse than the other two for the self-similar distribution (Figure 7), but is comparable on the Poisson distribution (Figure 8), where all techniques do quite well when the sample size is at least 256.

Figures 9 and 10 show results for text excerpts. Text is often well-modeled by a Zipf(1.0) distribution, so it is not surprising that the results are similar to that for the Zipf(1.0) data set (Figure 2). Likewise, results for words from the Brown corpus are similar (Figure 11), although sample-count is somewhat less reliable than with the other text excerpts.

Finally, the spatial data sets show similar results (Figures 12 and 13). The common relative order of the three algorithms holds, although sample-count does almost as bad as naive-sampling for these two data sets.

In summary, sample-count and tug-of-war are always clear winners, although in rare cases naive-sampling performs almost as well as either sample-count or tug-of-war. Both sample-count and tug-of-war perform well even with a very modest number of sample points relative to the data set sizes, reliably estimating self-join sizes of both synthetic and real-world data sets. In around half of the plots,

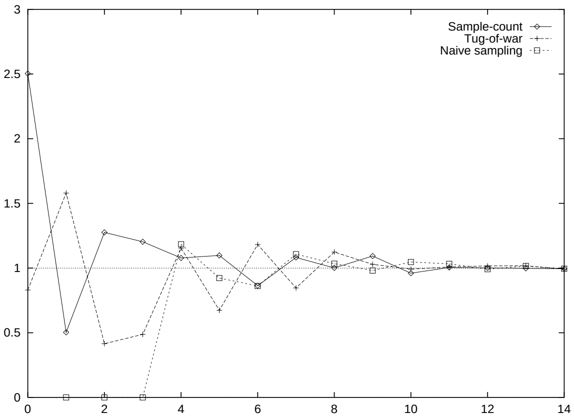

Figure 8: Poisson distribution

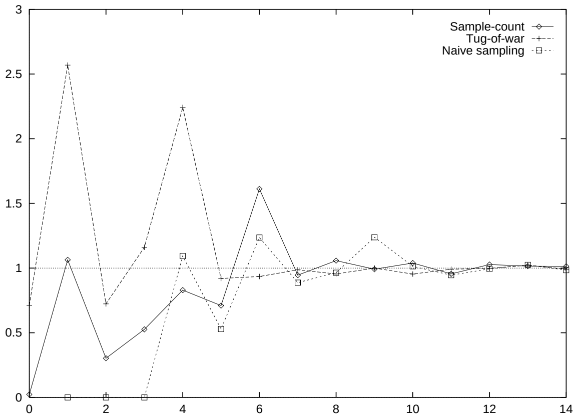

Figure 9: Wuthering Heights

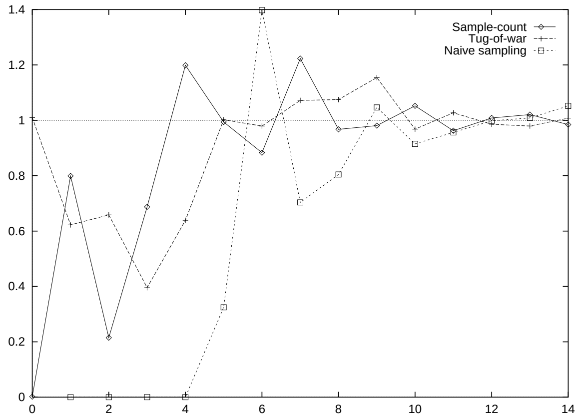

Figure 10: The book of Genesis

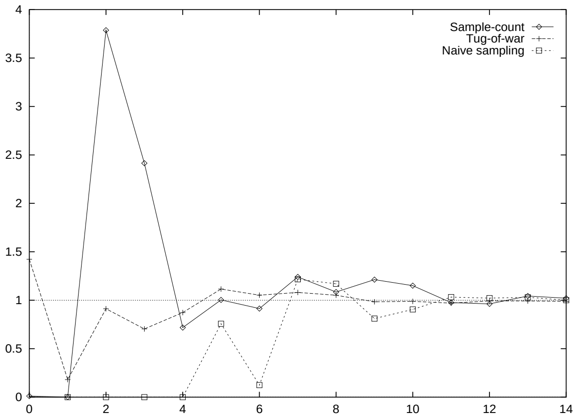

Figure 11: Excerpt from the Brown Corpus

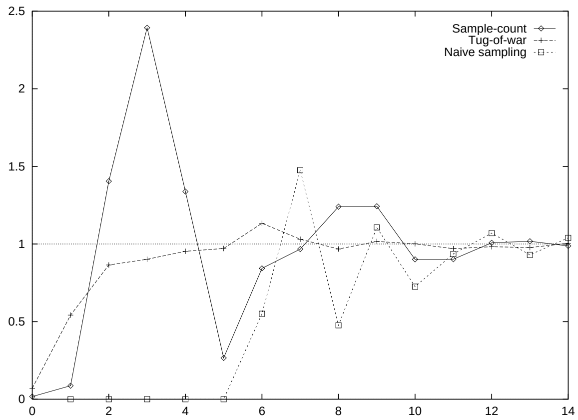

Figure 12: x-coordinates from a spatial point set

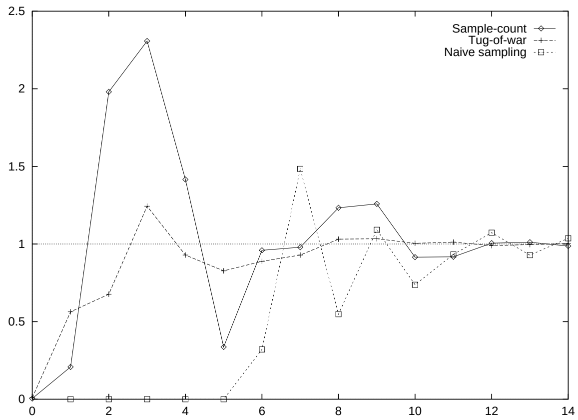

Figure 13: y-coordinates from a spatial point set

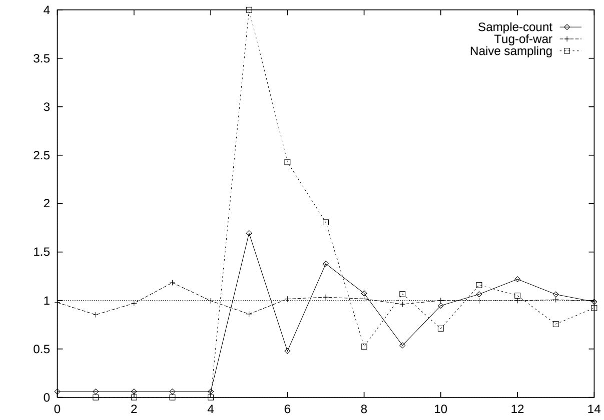

Figure 14: A pathological example

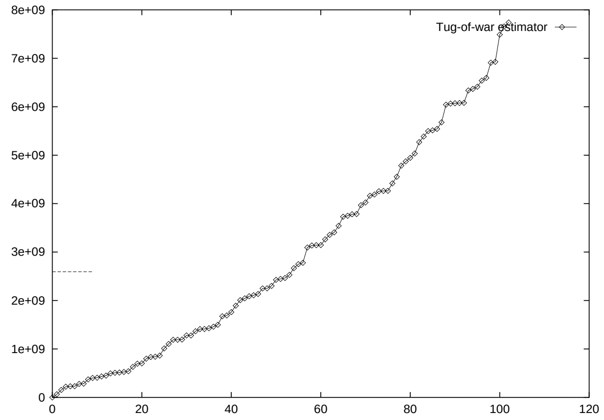

Figure 15: Robustness of estimators X_{ij}

the tug-of-war algorithm converges noticeably faster than the sample-count algorithm. For most of the remaining plots, the difference between the two is modest. The most dramatic case in which sample-count produces better estimates than tug-of-war is for the Uniform distribution.

Our results show that tug-of-war needed only 4–256 memory words, depending on the data set, in order to estimate the self-join size to within a 15% relative error; on average, this is over 4 times smaller sample size than needed by sample-count, and over 50 times smaller sample size than needed by naive-sampling.

3.2 Separating tug-of-war and sample-count

The relative closeness of sample-count and tug-of-war on the above data sets contrasts with the large gap in the asymptotic bounds on sample size. Recall that in order to guarantee from Theorem 2.1 that for any input set, Algorithm sample-count produces an accurate estimate with high probability, a $\Theta(\sqrt{t})$ sample size is needed. In contrast, Theorem 2.2 reveals that only $O(1)$ memory words are needed for Algorithm tug-of-war to produce an accurate estimate with high probability for any input set. In order to verify that this worst case gap can occur (and is not simply an artifact of the upper bound analysis), we construct a pathological data set for which sample-count converges particularly slowly, in contrast to tug-of-war. In the "path" data set, 40000 values occur exactly once, and one value occurs 800 times. The estimates for this pathological data set are displayed in Figure 14, and indeed the performance closely matches the theoretical prediction.

3.3 Robustness of tug-of-war estimates

Another approach to measuring the reliability of the tug-of-war estimator is to consider the distribution of the individual estimators X_{ij}.[4] In Figure 15, we plot 103 individual estimators for the zipf1.5 data set in Table 1. The estimators have been sorted in increasing order. The value of the estimator is plotted as a function of the estimator number. The actual self-join size is depicted by a dashed horizontal line segment extending from the y-axis. Note that the median individual estimator is slightly below the

[4] Recall that the overall estimator is obtained by computing averages of groups of these individual estimators, and then taking the median of the group averages. Thus we expect these individual estimators to have much larger variance than our overall estimator.

actual self-join size, but that many of the overestimates incur a larger absolute error than the worst of the underestimates. The most telling observation is the lack of clustering around the actual self-join size: the estimators are fairly equally spread across the range. This indicates why taking averages, or medians of averages, of the individual estimators $X_{i,j}$ is so essential to getting high-quality estimates.

4 Signature Schemes for Join Size Estimation

In this section, we study signature schemes for join size estimation. The goal is to maintain a small signature for each relation independently such that at any point we can estimate the join size of any two relations. In the traditional approach of join size estimation without the benefit of precomputed signatures, it is well-known that join size estimation is ineffective when the join size to be estimated is small. Thus previous work on estimating join sizes has advocated the use of "sanity bounds" [LN95, LNS90]: the goal is to develop procedures that provide an accurate estimate whenever the join size is at least B and otherwise report that the join size is less than B, and to minimize the B. (Typical values for B are $n^{3/2}$ or $n \log n$.) Sanity bounds are appropriate for join size estimation: there is a strong motivation to estimate the join size accurately only when the join size is large, since in such cases the resources that would be consumed to perform the join are large.

We consider join size estimation in the presence of an a priori sanity (lower) bound on the join size and present the first results showing that the simple random sampling approach has essentially the best estimation guarantees (worst case guarantees, over all possible relations) among all possible signature schemes. Since the estimation guarantees are not satisfactory, we propose a more refined analysis that takes into account the self-join sizes of the participating relations. We assume now two bounds: a lower bound on the join size and an upper bound on the self-join size. We present a signature scheme that gives provably better join size estimation for many settings of these two parameters. This algorithm is based on the tug-of-war approach outlined in Section 2.2.

4.1 Analysis of random samples as signatures

First we study the simple signature scheme of randomly selecting each tuple from a relation with probability p, and storing the value of the joining attribute for that tuple as the signature for the relation. To estimate the join size of two relations F and G, we compute the size of the join of their signatures and scale the result by p^{-2}. (This procedure is called *t_cross* in [HNSS93].)

We can view the tuples in F and G as nodes in the two sides of a bipartite graph $\Gamma = (\Gamma_V, \Gamma_E)$. There is an edge between a node $f \in F$ and a node $g \in G$ if and only if tuples f and g have the same value on the joining attribute. Then $|\Gamma_E| = |F \bowtie G|$, the join size of F and G. The join size of their samples is the number of edges spanned in Γ by the nodes in the samples.

Lemma 4.1 *Let Γ be any graph on n nodes. Assume we select nodes of Γ randomly, each with probability $p \geq \frac{1}{n}$. Let X denote the random variable whose value is the number of edges that are spanned by the nodes in the sample. Then $E(X) = |\Gamma_E|p^2$ and $\mathrm{Var}(X) \leq |\Gamma_E|p^2 + \sum_{i=1}^{n} d_i^2 p^3$, where d_i is the degree of node i in Γ.*

Since $\sum_{i=1}^{n} d_i^2 \leq n \sum_{i=1}^{n} d_i = 2n|\Gamma_E|$, we can bound $\mathrm{Var}(X)$ in Lemma 4.1 by $3n|\Gamma_E|p^3$. Note that if $E(X)^2 \geq \alpha \mathrm{Var}(X)$ for a constant $\alpha > 1$, we can apply the Chebychev inequality to obtain a (small) constant factor error with (high) constant probability. $\mathrm{Var}(X) \leq E(X)^2/\alpha$ if $3|\Gamma_E|np^3 \leq |\Gamma_E|^2p^4/\alpha$, i.e., $p \geq 3\alpha n/|\Gamma_E|$. This shows that a sample of expected size $np > 3\alpha n^2/|F \bowtie G|$ is sufficiently large.

We conclude:

Lemma 4.2 *Suppose we have an a priori lower bound B on the join size. The simple sampling signature scheme estimates the join size with constant relative error with high probability if the random sample has size at least cn^2/B, for a constant $c > 3$ determined by the desired accuracy and confidence.*

Note that random samples of each relation can be maintained incrementally with small overheads as new data is inserted or deleted into the relation [Vit85, GMP97], and hence one can track join sizes in $O(n^2/B)$ memory words using this approach.

4.2 Lower bounds on signature schemes for join size estimation

We prove that, to within constant factors on the signature size, the simple sampling algorithm in the previous subsection cannot be improved (measured by worst case analysis) given no further assumptions. The lower bound applies to all possible signature schemes, including static signatures that may or may not have efficient incremental maintenance.

We say an estimate is "good with high probability" if it is within a 1% relative error with 99% probability.

Theorem 4.3 *Let Φ be any scheme which assigns bit strings to database relations, so that there is a random or deterministic pairing function D such that given two relations F and G of size n the formula $D(\Phi(F), \Phi(G))$ gives a good estimate on the join size of F and G with high probability, when an a priori lower bound B, $n \leq B \leq n^2/2$, is given on the join size. Then the length of the bit string that Φ assigns to relations of size n must be at least $(n - \sqrt{B})^2/B$.*

Proof. Let $m = n - \sqrt{B}$. Define $t = 10m^2/B$ and fix a set T of t possible values for the joining attribute, denoted *types*. Let D_1 be the uniform probability distribution on uni-type relations over T; namely, with probability $1/t$ we select the relation comprising m tuples of type i, where $1 \leq i \leq t$. We define another distribution D_2 in the following way: Let $\mathcal{S}$ be a family of subsets of $\{1, 2, \ldots, t\}$ such that: (1) All sets in $\mathcal{S}$ have size $m^2/B = t/10$. (2) $|\mathcal{S}| = 2^{m^2/B} = 2^{t/10}$. (3) For all $S_1, S_2 \in \mathcal{S}$, $S_1 \neq S_2$, we have $|S_1 \cap S_2| \leq m^2/2B = t/20$. One can show the existence of such a set system using the probabilistic method. For each $S \in \mathcal{S}$, we define a relation S^* of size m comprising B/m tuples of each type in S. Let $\mathcal{S}^*$ be the set of relations so defined. We define D_2 to be the uniform distribution on relations in $\mathcal{S}^*$.

To ensure that all join sizes are at least B, we augment each relation in D_1 and D_2 to also have $\sqrt{B}$ tuples of type 0. Thus the total size of each relation is n.

Let F be a relation randomly chosen from D_1 and let G be a relation randomly chosen from D_2. The join size of F and G is either B or $B + m(B/m) = 2B$. Applying Yao's standard technique, it suffices to show that any deterministic scheme that assigns strings of length at most $(m^2/B) - 1$ fails to estimate the join size with small error with probability bounded away from 0 for a random pair $F \in D_1, G \in D_2$. Consider partitioning the relations into classes according to the bit string assigned them by Φ. For each relation in D_1, the pairing function gives the same estimate for all relations in D_2 in the same class. However, for each class, there can be at most one relation in D_2

for which the estimate has less than 50% error for more than 95% of the relations in D_1. To see this, consider $S_1, S_2 \in \mathcal{S}$ such that the corresponding relations in D_2 map to the same class, and let $T' = \{t \in (S_1 - S_2) \cup (S_2 - S_1)\}$. For each D_1 whose type is in T', the join size is B for one of S_1 and S_2 and $2B$ for the other; thus any estimate will have at least 50% error for at least one of them. By the properties of $\mathcal{S}$, we have $|T'| \geq 2(t/10 - t/20) = t/10$, and hence for one of them, the estimate will have at least 50% error for more than $t/20 = 5\%$ of the relations in D_1. Since the number of distinct bit strings is at most $2^{m^2/B}/2$, we get that for a constant fraction of the pairs $F \in D_1, G \in D_2$ the scheme fails to estimate the join size with small error. ∎

Thus if B is $o(n^2)$, then the bit strings must be at least $n^2/(1+o(1))B$ long. Comparing Lemma 4.2 and Theorem 4.3, we have that (i) the sampling signature scheme with an expected $\Theta(n^2/B)$ values stored is good with high probability, and (ii) no signature scheme is good with high probability unless it has $\Omega(n^2/B)$ bits stored.

This lower bound implies estimation guarantees that are not satisfactory in many cases. Thus in the next subsection, we propose a more refined analysis that takes into account the self-join sizes of the participating relations. We assume now two bounds: a lower bound on the join size and an upper bound on the self-join size, and ask if in this case, can one do better than random sampling? We show that indeed one can do better by presenting a signature scheme that gives provably better join size estimation for many settings of these two parameters.

4.3 The tug-of-war join signature scheme

Recall that our goal is to maintain a small signature for each relation independently such that at any point we can estimate the join size of any two relations. Our new signature scheme is based on tug-of-war signatures, and provides guarantees on join size estimation as a function of the self-join sizes of the joining relations. Specifically, the scheme gives an estimator for the join size of any two relations F and G whose error is (with high probability) at most $\sqrt{2 \cdot \mathrm{SJ}(F) \cdot \mathrm{SJ}(G)}$, where $\mathrm{SJ}(F)$ and $\mathrm{SJ}(G)$ are the self-join sizes of F and G. The signature that enables this estimator for any two relations is only $O(1)$ memory words per relation. Using this signature as a building block, we construct a larger signature of k memory words comprising k independent signatures per relation. An estimator based on taking the arithmetic mean of the k individual estimators reduces the error by a factor of $\sqrt{k}$.

Let $D = \{1, 2, \ldots, t\}$ be the domain of the joining attribute. Let F and G be two relations of n tuples each. For $i = 1, \ldots, t$, let f_i and g_i be the number of tuples in F and G whose joining attribute value is i. The join size $|F \bowtie G| = \sum_{i=1}^{t} f_i \cdot g_i$.

Let $\{\epsilon_i\}_{i=1}^{t}$ be four-wise independent $\{-1, 1\}$-valued random variables. For F and G we create the signatures $S(F) = \sum_{i=1}^{n} \epsilon_i f_i$ and $S(G) = \sum_{i=1}^{n} \epsilon_i g_i$, respectively.

The estimator for $|F \bowtie G|$ is simply $S(F) \cdot S(G)$.

Lemma 4.4 *Let $S(F)$ and $S(G)$ be tug-of-war join signatures for relations F and G. Then*

$$\begin{aligned} \mathrm{E}(S(F) \cdot S(G)) &= |F \bowtie G| \qquad (1) \\ \mathrm{Var}(S(F) \cdot S(G)) &\leq 2 \cdot \mathrm{SJ}(F) \cdot \mathrm{SJ}(G), \qquad (2) \end{aligned}$$

where $\mathrm{SJ}(F)$ and $\mathrm{SJ}(G)$ are the self-join sizes of F and G.

Proof.

$$\begin{aligned}
\mathrm{E}(S(F)\cdot S(G)) &= \mathrm{E}(\sum_{i=1}^{t}\epsilon_i^2 f_i g_i + \sum_{1\le i\ne j\le t}\epsilon_i\epsilon_j f_i g_j) \\
&= \sum_{i=1}^{t} f_i g_i = |F \bowtie G|,
\end{aligned}$$

since $\mathrm{E}(\epsilon_i\epsilon_j) = 0$ for $1 \le i \ne j \le t$. To prove Equation (2) define

$$X = S(F)\cdot S(G) - \mathrm{E}(S(F)\cdot S(G)) = \sum_{1\le i\ne j\le t}\epsilon_i\epsilon_j f_i g_j.$$

Since $\mathrm{E}(X^2) = \mathrm{Var}(S(F)\cdot S(G))$, we have:

$$\mathrm{Var}(S(F)\cdot S(G)) = \sum_{1\le i\ne j\le t} f_i^2 g_j^2 + \sum_{1\le i\ne j\le t} f_i g_i f_j g_j. \tag{3}$$

Now from

$$\sum_{1\le i\ne j\le t} f_i^2 g_j^2 = \sum_{1\le i\le t} f_i^2 \sum_{1\le j\le t} g_j^2 - \sum_{1\le i\le t} f_i^2 g_i^2,$$

and

$$\begin{aligned}
\sum_{1\le i\ne j\le t} f_i g_i f_j g_j &= \left(\sum_{1\le i\le t} f_i g_i\right)^2 - \sum_{1\le i\le t} f_i^2 g_i^2 \\
&\le \sum_{1\le i\le t} f_i^2 \sum_{1\le j\le t} g_j^2 - \sum_{1\le i\le t} f_i^2 g_i^2,
\end{aligned}$$

and Equation (3), we conclude that

$$\begin{aligned}
\mathrm{Var}(S(F)\cdot S(G)) &\le 2\left(\sum_{1\le i\le t} f_i^2 \sum_{1\le j\le t} g_j^2 - \sum_{1\le i\le t} f_i^2 g_i^2\right) \\
&\le 2\cdot \mathrm{SJ}(F)\cdot \mathrm{SJ}(G).
\end{aligned}$$

■

Note that the tug-of-war signature scheme described in this section is a better join size estimator than the random sample estimator, because already it is a better estimator for the self-join (as demonstrated earlier in this paper – see Lemma 2.3).

The estimation guarantees of the tug-of-war signature scheme can be enhanced by repeating the basic scheme $k > 1$ times and taking the arithmetic mean of the results. We denote this scheme by k-TW. The signature size of the k-TW is k memory words per relation.

Theorem 4.5 *Let F and G be two relations such that $|F \bowtie G| \ge B_1$, $\mathrm{SJ}(F) \le B_2$, and $\mathrm{SJ}(G) \le B_2$. Then the k-TW estimator with*

$$k = \frac{c\cdot \mathrm{SJ}(F)\cdot \mathrm{SJ}(G)}{B_1^2} \le \frac{cB_2^2}{B_1^2}$$

estimates $|F \bowtie G|$ within constant relative error with high probability, for a constant $c > 2$ determined by the desired accuracy and confidence.

Proof. By Lemma 4.4, the variance of the 1-TW estimator is upper bounded by $2 \cdot \mathrm{SJ}(F) \cdot \mathrm{SJ}(G) \leq 2B_2^2$. Since the k-TW estimator is the arithmetic mean of k independent 1-TW estimator, we can upper bound its variance by $2 \cdot \mathrm{SJ}(F) \cdot \mathrm{SJ}(G)/k \leq 2B_2^2/k$. We also have a B_1^2 lower bound on the square of the expectation. The theorem follows from the Chebychev inequality. ∎

Note that for each 1-TW, the $\{\epsilon_i\}_{i=1}^t$ can be determined by selecting at random from a family of 4-wise independent hash functions. Thus for k-TW, we select independently at random k such hash functions. Let Z_i be the signature for the ith hash function h_i. For each insertion into the relation of a new tuple with joining attribute value x, for $i = 1, \ldots, k$, we add $h_i(x)$ $(= 1$ or $-1)$ to Z_i; for each deletion from the relation of an existing tuple with joining attribute value x, we subtract $h_i(x)$ from Z_i. Thus we can use k-TW signatures to track join sizes in limited storage (namely k memory words per relation).

Note that k-TW does not require a priori knowledge about the length of the sequence, the size of either relation at query times, or the number of distinct values in either relation.

A remark on signatures for a priori join pairs. We have considered in this paper the set-up in which the signature for an individual relation F is computed in isolation and must provide good quality estimates for $|F \bowtie G|$ for any other relation G. This rules out adapting approaches used in traditional join size estimation that supplement sampling in one relation with indexed lookups of the number of tuples with a joining attribute value in the other relation, such as the adaptive sampling of [LN95] and the bifocal sampling of [GGMS96] (procedures with indexed lookups are called *t_index* in [HNSS93]). An alternative scenario to consider is to be given a set of join pairs and compute a signature for each pair, and to incrementally maintain these signatures. The practical problem then is that the size of the signatures and the work for incremental maintenance may scale with the number of pairs. For example, the construction in the lower bound of Theorem 4.3 shows that large signatures are required to obtain good estimates with high probability, even when restricting the set of joins to be relations from D_1 joining with relations from D_2.

4.4 Analytical Comparison of the Two Algorithms

In this subsection, we analytically compare the memory words needed by the random sampling approach and by the tug-of-war approach, in order to achieve constant relative error with high probability. By Lemma 4.2, we have that the random sampling approach uses $\Theta(\frac{n^2}{B})$ memory words, where n is the size of each relation, and B is the sanity bound, $n \leq B \leq \frac{n^2}{2}$. By Theorem 4.5, we have that k-TW uses $O(\frac{C^2}{B^2})$ memory words, where C is an upper bound on the size of the self-join for both relations. Ignoring constants, it follows that k-TW improves upon the sampling approach when $\frac{C^2}{B^2} < \frac{n^2}{B}$, i.e., when $C < n\sqrt{B}$.

It is interesting to note that the self-join sizes for many of the data sets in Table 1 are indeed smaller than $n\sqrt{B}$ for modest B. For the uniform, mf3, and path data sets, k-TW is better even for $B = n$, and the advantage is about 1000, 20, and 150, resp. For others, in order for k-TW to have an advantage, B needs to be larger than n by roughly a factor of 6700 for selfsimilar, 4000 for zipf1.5, 500 for poisson, 150 for zipf1.0, 50 for brown2, and 1-10 for mf2, wuther, genesis, xout1, and yout1.

In short, for very uniform situations, k-TW has a significant advantage because C is small. For highly-skewed distributions, C is rather high, so k-TW is expensive unless B is sufficiently large compared to n (a factor of 1000, where n is about 100,000). For text data sets, the situation is intermediate (modest skew), and k-TW works better for moderately small B and above.

5 Conclusions

This paper has considered the problem of tracking (approximate) join and self-join sizes in limited storage in the presence of insertions and deletions to the relations. The goal is to maintain a small synopsis of the data in each relation, kept up-to-date as the data changes, in order to provide a high quality estimate of a join or self-join size, on demand at any time.

For self-joins, we discuss three algorithms, sample-count, tug-of-war, and naive-sampling, focusing on extensions to handle deletions and on experimental evaluation. Extending the results in [AMS99], we present analytical bounds demonstrating that, for the same size synopsis, tug-of-war is more accurate than sample-count which is more accurate than naive-sampling. Our experimental results on a variety of synthetic and real-world data sets support this relative ordering in accuracy, although the gap between tug-of-war and sample-count is often small, and indeed, sometimes sample-count is more accurate. The naive-sampling algorithm, on the other hand, does considerably worse than the other two.

For joins, our goal is to maintain a small synopsis (a join signature) of each relation such that join sizes can be accurately estimated between any pairs of relations. We show that taking uniform random samples for join signatures can lead to inaccurate estimation unless the sample size is quite large, namely $\Theta(n^2/B)$, where n is the size of each relation and B is an a priori sanity lower bound on the join size ($n \leq B \leq \frac{n^2}{2}$). Moreover, we prove a lower bound that shows that no signature scheme can provide good estimation guarantees unless it stores $\Omega(n^2/B)$ bits. Thus no other scheme can significantly improve upon random sampling without further assumptions. Finally, we present a signature scheme based on tug-of-war signatures that provides guarantees on join size estimation as a function of the self-join sizes of the joining relations. This scheme can significantly improve upon the sampling scheme across a range of self-join sizes whenever the self-join sizes are smaller than $n\sqrt{B}$. Moreover, the join signature for a relation can be maintained incrementally in the presence of insertions and deletions to the relation.

A possible concern for tracking algorithms is the cost they occur at the time the data is updated. In a typical (offline) data warehouse scenario, data loading occurs in batch mode, in between batches of queries; the tracking algorithms described in this paper are well-suited for such scenarios. On the other hand, in scenarios where data updates occur intermixed with queries, tracking algorithms must have very low overhead in order to avoid creating a concurrency bottleneck: Even constant time per update may be too slow. An alternative is periodically run the tracking algorithm in batch mode, by stepping through any additions to the update log since the previous run. In such cases, the accuracy guarantees must be weakened accordingly to account for updates not yet processed by the tracking algorithm.

Future work includes performing an experimental study of the tug-of-war join signature scheme ro complement our analytical comparison, and extending the work to more general scenarios such as three-way joins.

Acknowledgements

The first author is supported in part by a USA-Israel BSF grant and by the Fund for Basic Research administered by the Israel Academy of Sciences. The third author is supported in part by an Alon Fellowship, by a Tel Aviv University Grant, by the Israel Science Foundation founded by The Academy of Sciences and Humanities, and by the Israeli Ministry of Science. Part of this work was done while the second author was with Bell Labs and the fourth author was with AT&T Labs.

We thank Ken Church and Christos Faloutsos for providing some of the data sets used in our experimentations.

References

[AGP00] 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.

[AGPR99] 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.

[AMS99] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. of Computer and System Sciences*, 58:137–147, 1999.

[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.

[CCMN00] 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.

[CDN01] 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.

[CGRS00] 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.

[GGMS96] S. Ganguly, P. B. Gibbons, Y. Matias, and A. Silberschatz. Bifocal sampling for skew-resistant join size estimation. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 271–281, June 1996.

[Gib01] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *Proc. 27th International Conf. on Very Large Data Bases*, pages 541–550, September 2001.

[GKS01] A. C. Gilbert, Y. Kotidis, and M. J. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proc. 27th International Conf. on Very Large Data Bases*, pages 79–88, September 2001.

[GLR00] 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.

[GM98] 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.

[GM99] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In J. M. Abello and J. S. Vitter, editors, *External Memory Algorithms*, pages 39–70. AMS, 1999. DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Vol. 50. A two page summary appeared as a short paper in SODA'99.

[GMP97] 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.

[Goo89] I. J. Good. Surprise indexes and p-values. *J. Statistical Computation and Simulation*, 32:90–92, 1989.

[HH99] 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.

[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.

[HNSS93] P. J. Haas, J. F. Naughton, S. Seshadri, and A. N. Swami. Fixed-precision estimation of join selectivity. In *Proc. 12th ACM Symp. on Principles of Database Systems*, pages 190–201, May 1993.

[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.

[HÖT88] 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.

[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.

[IP99] 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.

[LM01] 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.

[LN95] R. J. Lipton and J. F. Naughton. Query size estimation by adaptive sampling. *J. Computer and System Sciences*, 51(1):18–25, 1995.

[LNS90] R. J. Lipton, J. F. Naughton, and D. A. Schneider. Practical selectivity estimation through adaptive sampling. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 1–12, May 1990.

[MVW00] Y. Matias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *Proc. 26th International Conf. on Very Large Data Bases*, pages 101–110, September 2000.

[Olk93] F. Olken. *Random Sampling from Databases.* PhD thesis, Computer Science, U.C. Berkeley, April 1993.

[Poo97] V. Poosala. *Histogram-based Estimation Techniques in Databases.* PhD thesis, Univ. of Wisconsin-Madison, 1997.

[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.

[VW99] 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.