Tracking Join and Self-Join Sizes in Limited Storage

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

Phillip B. Gibbons
Information Sciences Research
Bell Laboratories
Murray Hill, NJ
gibbons@research.bell-labs.com

Yossi Matias
Dept of Computer Science
Tel Aviv University
Tel Aviv, Israel
matias@math.tau.ac.il

Mario Szegedy
Information Sciences Research
AT&T Labs
Florham Park, NJ
ms@research.att.com

Abstract

Query optimizers rely on fast, high-quality estimates of result sizes in order to select between various join plans. Self-join sizes of relations provide bounds on the join size of any pairs of such relations. It also indicates the degree of skew in the data, and has been advocated for several estimation procedures. Exact computation of the self-join size requires storage proportional to the number of distinct attribute values, which may be prohibitively large. In this paper, we study algorithms for tracking (approximate) self-join sizes in limited storage in the presence of insertions and deletions to the relations. Such algorithms detect changes in the degree of skew without an expensive recomputation from the base data. We show that an algorithm based on a tug-of-war approach provides a more accurate estimation than one based on a sample-and-count approach which is in turn more accurate than a sampling-only approach.

Next, we study algorithms for tracking (approximate) join sizes in limited storage; the goal is to maintain a small signature of each relation such that join sizes can be accurately estimated between any pairs of relations. We show that taking random samples for join signatures can lead to inaccurate estimation unless the sample size is quite large; moreover, by a lower bound we show, 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 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.

1 Introduction

The skew of a data set represents how far the frequency distribution of the items that occur in the data set is from being uniform. The 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 indicates the degree of skew in the distribution of attribute values. For a relation A, the self-join size (also called the second frequency moment) on an attribute in A with value domain D is $\sum_{i\in D} a_i^2$, where a_i is the frequency of attribute value i in A. 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 provide bounds on the join size of any pairs of such relations, as follows. Consider the join of relations A and B on joining attribute(s) with value domain D. For $i \in D$, let a_i and b_i be the frequency of the ith value in A and B, respectively. Then the join size, $|A \bowtie B| = \sum_{i\in D} a_i b_i$, satisfies

$$|A \bowtie B| \leq \frac{\mathrm{SJ}(A) + \mathrm{SJ}(B)}{2},$$

where $\mathrm{SJ}(A) = |A \bowtie A|$ and $\mathrm{SJ}(B) = |B \bowtie B|$ are the self-join sizes on the joining attributes. To see this, 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 $\sum_{i\in D} a_i b_i \leq \sum_{i\in D}(a_i^2 + b_i^2)/2 = (\sum_{i\in D} a_i^2 + \sum_{i\in D} b_i^2)/2 = (\mathrm{SJ}(A) + \mathrm{SJ}(B))/2$.

For many distributions, such as zipfian and exponential, the self-join size uniquely determines the parameter of the distribution. For example, consider an exponential distribution, in which the ith most popular value occurs with frequency $n(\alpha-1)\alpha^{-i}$ in a relation, A, of size n. Then $\mathrm{SJ}(A) = \sum_i (n(\alpha-1)\alpha^{-i})^2 = n^2(\alpha-1)^2 \sum_i (\alpha^2)^{-i} = n^2(\alpha-1)^2/(\alpha^2-1) = n^2(\alpha-1)/(\alpha+1)$. It follows that $\alpha = (n^2 + \mathrm{SJ}(A))/(n^2 - \mathrm{SJ}(A))$.

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.

In this paper, we study algorithms for tracking (approximate) self-join sizes in limited storage in the presence of insertions and deletions to the database. Alon *et al* [AMS96] proposed two algorithms for tracking self-join sizes in the presence of insertions, which we denote as *sample-count* and *tug-of-war*, and presented upper bounds on the space required to guarantee a desired accuracy with high probability. We consider the practical aspects of these algorithms, by considering also deletions, implementation issues, and experimental evaluation, comparing these two approximation algorithms to a naive sampling approach, across a range of data sets. Our experiments demonstrate the practical utility of the proposed algorithms, by showing that good estimates are obtained while using only a small fraction of the memory required for an exact self-join size. We compare the accuracy of the three approximation algorithms, demonstrating that unless the self-join size is predominantly determined by very few items, the naive sampling approach may not be very useful. In contrast, both approximation algorithms presented by Alon *et al* provide accurate estimations. 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*.

Next, we study algorithms for tracking (approximate) join sizes in limited storage; the goal is to maintain a small signature of each relation such that join sizes can be accurately estimated between any pairs of relations. We show that taking random samples for join signatures can lead to inaccurate estimation unless the sample size is quite large. Moreover, by a lower bound we show, no other signature scheme can provide significantly better estimation guarantees without further assumptions. These negative results are shown to hold even in the presence of sanity bounds.[1] 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.

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

Synopsis data structures and tracking algorithms. The signature schemes studied in this paper are examples of *synopsis data structures*, data structures whose size is substantively smaller than the full data set and provide typically approximate answers to queries. There are many existing examples of synopsis data structures [BDF+97, GM98b]. In brief, a synopsis data structure has the following advantages over a non-synopsis (e.g., linear space) data structure: (a) it may reside in main memory, enabling query responses and data structure updates that avoid disk accesses altogether, (b) it can be transmitted remotely at minimal cost, (c) it has minimal impact on the overall storage costs of a system, (d) it leaves space in the memory for other processing (available main memory is a precious resource for external memory algorithms), and (e) it can serve as a small surrogate for data sets that are currently expensive or impossible to access. On the other hand, the answers are typically only approximate, not exact. This is acceptable in many cases, such as the scenario considered in this paper of size estimation within a query optimizer.

One can consider synopsis data structures that are static or dynamic (i.e., incrementally maintained in the presence of data insertions and deletions). *Tracking in limited storage* considers this latter case. Tracking algorithms can detect changes in the quantity to be estimated without an expensive recomputation from the base data, and can also be used to compute an (approximate) answer/estimation in one pass and limited storage. On the other hand, they incur a cost 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; tracking algorithms can be well-suited for such scenarios. In scenarios where data updates occur intermixed with queries, the tracking algorithm must have very low overhead in order to avoid creating a concurrency bottleneck, or otherwise must be applied periodically in batch mode. In this latter case, the accuracy guarantees are weakened accordingly to account for updates not yet propagated to the tracking algorithm.

We view the results in this paper as a step towards the further understanding and study of synopsis data structures and tracking algorithms.

Related work. [BDF+97] presents a survey of *data reduction* techniques for massive data sets. [GM98b] presents a formal framework for evaluating 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, BDF+97, GMP97a, GMP97b, HHW97, GM98a, AGPR99, HH99, AGP99, MS99]). The work in [HHW97, AGPR99, HH99] has looked at the problem of providing approximate answers to queries seeking aggregates (e.g., sum, avg) of attribute values for the tuples satisfying a predicate that occur in the join of multiple relations. Thus although joins are involved, the goal in these works is to estimate the aggregate, not the join size.

There is an extensive literature on join size estimation (e.g., [HÖT88, LNS90, HNSS93, LN95, GGMS96]). These papers consider the traditional approach of estimating the join sizes without the benefit of precomputed signatures, and hence 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 the relation. (Such histograms can be maintained incrementally using the algorithm in [GMP97b].) However, there are no good guarantees on the accuracy of such estimations. Manku *et al* [MRL98] presented tracking algorithms for computing approximate medians and other quantiles in limited storage.

Outline. The rest of the paper is organized as follows. In Section 2 we describe the *sample-count* and *tug-of-war* algorithms, implementation issues for both algorithms, and extensions to handle deletions. We also present a new lower bound for the naive sampling approach. Section 3 presents our experimental study of the three algorithms for self-join estimation. Section 4 presents our new results for join size estimation. Finally, concluding remarks appear in Section 5.

[1] *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 considerably the estimation guarantees.

2 Tracking self-join sizes

In this section we describe the two algorithms for approximating self-join sizes in limited storage presented in [AMS96]. For each algorithm, we provide extensions to handle deletions and present trade-offs in implementing the basic steps of algorithm. Let $A = (v_1, v_2, \ldots, v_n)$ 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 algorithms is a natural one. In order to estimate the self-join size, $\mathrm{SJ}(A)$, a random variable is defined that can be computed under a given space constraint, whose expected value is $\mathrm{SJ}(A)$, 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.

2.1 Algorithm *sample-count*

The number of memory words used by the algorithm is $s = s_1 \cdot s_2$, where s_1 is a parameter that determines the accuracy of the result, and s_2 determines the confidence. The algorithm computes s_2 random variables $Y_1, Y_2, \ldots, Y_{s_2}$ and outputs their median Y. Each Y_i is the average of s_1 random variables $X_{ij} : 1 \le j \le s_1$, where the X_{ij} are independent, identically distributed random variables. Each of the variables $X = X_{ij}$ is computed from the sequence in the same way as follows:

- Choose a random member v_p of the sequence A, where the index p is chosen randomly and uniformly among the numbers $1, 2, \ldots, n$; suppose that $v_p = l \ (\ \in D)$.
- Let $r = |\{q : q \ge p, v_q = l\}| \ (\ \ge 1)$ be the number of occurrences of l among the members of the sequence A following v_p (inclusive).
- Let $X = n(2r-1)$.

Extensions. Note that in the tracking scenario, the sequence A is observed as a series of insertions, and we may be required at any point to answer a self-join size query on the sequence to date. Moreover, the length, n, of the sequence is not fixed in advance, but is increasing with each insert.

We can adapt this algorithm (particularly the first step) to handle the tracking scenario, as follows. We start with $n = 1$, select v_1 as our random member, and set r to be 1. In general, after $n-1$ inserts, we have (for each variable X_{ij}) some value for our random member v_l and for r. When the next element v_n is inserted, we replace v_l by that element with probability $1/n$. In case of such a replacement, we reset r to be 1. If no replacement, v_l stays as it is, and r increases by 1 if $v_n = v_l$ and otherwise does not change. The cost of adapting the s sample points is $O(s)$, and this correction process may be too expensive if executed for every new sequence member. A more efficient implementation avoids the adaptation after every insertion using standard techniques that trade off correction frequency versus estimation effectiveness between corrections.

For the implementation of the second step, we use the following approach in order to avoid incrementing k counters each time a value v is inserted that occurs k times among the s selected sample points (large k will be expected for highly skewed data). For each value v in the (current) sample, we maintain a count k_v of the number of sample points with value v and an aggregate counter, C_v, corresponding to the sum of the k_v r-counters associated with sample points with value v. For each sample point, we also store the value of C_v at the time the sample point was inserted. The values k_v and C_v are stored in a lookup table using v as the lookup key. On the arrival of a new sequence member with value v, we retrieve k_v and C_v, and increment C_v by k_v. If the new member is selected to be in the sample, then we also increment k_v and store the value of C_v with the sample point. This results in $O(1)$ time with high probability to process the new insert, regardless of the input set and of the sample size s. Note that the individual r-counters are not kept. When they are needed in order to produce an estimate, the k_v counters for a value v are calculated in $O(k_v)$ time by reversing the steps used to generate C_v.

To handle deletions, we assume that the adversary cannot adapt the sequence in response to the random choices made by our algorithm. We first observe that for the purpose of our estimation algorithms, we can replace each sequence member by its value (so that sequence members with the same value are indistinguishable). Thus, whenever there is a deletion with value v, 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 nill 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 nill value, and in addition we find the nearest member to the left of it with value v and replace it with a nill value as well. The non-nill 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 nill values are ignored.

We now show how the fast implementation of the second step of *sample-count* can be extended to handle deletions as well. In response to a *delete*(v), we reverse the operations that were done when the last remaining member with value v was inserted. If the value v is in the sample (which can be determined by table lookup), we retrieve k_v and C_v, and decrement C_v by k_v. If C_v is now smaller than one of the "C_v at time selected", then remove that sample point and decrement k_v, since we know that the member was selected into the sample upon the occurrence of the value v which is now deleted. This results in $O(1)$ time with high probability to process the new delete. Moreover, we have reduced the scenario with deletions to one with only insertions, and we can immediately apply the corresponding theorem in [AMS96], to obtain:

Theorem 2.1 *The estimate Y computed by the above algorithm satisfies:*

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

Note that we handle deletions as they occur, since in the tracking scenario of this paper, we must be prepared at all times to provide an answer to self-join size estimation queries on the sequence to date. Moreover, note that the *delete* operation may remove sample points without replacing them, dropping the number of sample points below s. 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.

Note that the *sample-count* algorithm is reminiscent of the algorithm in [GM98a] for maintaining "counting samples". 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 self-join size estimation that a value 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 the averaging and median techniques described above within the limited storage.

2.2 Algorithm *tug-of-war*

The *tug-of-war* algorithm can be illustrated as follows: Suppose that a crowd consists of several groups of varying numbers of people, and that our goal is to estimate the skew in the distribution of people to groups. That is, we would like to estimate $\mathrm{SJ}(A)$ for the set $\{v_i\}_{i=1}^n$, where v_i is the group to which the i'th person belongs. We arrange a tug-of-war, forming two teams by having each group assigned at random to one of the teams. Equating the displacement of the rope from its original location with the difference in the sizes of the two teams, it is shown in [AMS96] that the expected square of the rope displacement is exactly $\mathrm{SJ}(A)$, and that the variance is reasonably small.

In more detail, the number of memory words used by *tug-of-war* is $s = s_1 \cdot s_2$, where s_1 is a parameter that determines the accuracy of the result, and s_2 determines the confidence. As in *sample-count*, the output Y is the median of s_2 random variables $Y_1, Y_2, \ldots, Y_{s_2}$, each being the average of s_1 random variables $X_{ij} : 1 \leq j \leq s_1$, where the X_{ij} are independent, identically distributed random variables. Each $X = X_{ij}$ is computed from the sequence in the same way, as follows:

- Select at random a 4-wise independent mapping $i \mapsto \epsilon_i$, where $i \in \{1, 2, \ldots, t\}$ and $\epsilon_i \in \{-1, 1\}$.
- Let $Z = \sum_{i=1}^t \epsilon_i m_i$, where m_i is the number of members with value i.
- Let $X = Z^2$.

Extensions. To implement the first step, we need to select s independent hash functions, $h(v) = \epsilon_v \in \{-1, 1\}$, which can be done in $O(s)$ time. In practice it may be often reasonable to use hash functions that may not be 4-wise independent but easier to compute. In the second step, we maintain s program variables that hold the partial sums $Z = \sum_{j=1}^n h(v_j) = \sum_{j=1}^n \epsilon_{v_j}$, where n is the current sequence length. For each incoming sequence member with value i we compute the s independent mappings ϵ_i, and add them to the corresponding program variables Z in $O(s)$ time. To handle deletions, given an input sequence $\hat{A}$ as above, we imitate running algorithm *tug-of-war* on A by the following simple correction: In response to a *delete*(v), we reverse the operations that were done when the last remaining member with value v was inserted: for each program variable Z we subtract ϵ_v. It follows from the corresponding theorem in [AMS96] that:

Theorem 2.2 *The estimate Y computed by the above algorithm satisfies:*

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

2.3 Algorithm *naive-sampling*

We contrast algorithm *sample-count* and algorithm *tug-of-war* with the following naive sampling heuristic (not considered in [AMS96]), denoted below as algorithm *naive-sampling*. We sample s elements (without replacement) from the sequence, and compute the self-join size, $\mathrm{SJ}(S)$, 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 $\mathrm{SJ}(S)$ into an estimator X whose expected value is $\mathrm{SJ}(A)$:

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

We have the following lower bound on the sample size required to provide a good quality estimate of the self-join size. This lower bound applies even for static relations (i.e., the difficulty arises even when there is no tracking requirement).

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 F contain n items of different values. Let G contain $n/2$ pairs of items such that each pair contains items with the same value. Members of different pairs have different values. The estimator for F will be n. Since F and G are nearly indistinguishable to samples of size $o(\sqrt{n})$, the estimator for G will also be n with a sizable probability p. On the other hand, $\mathrm{SJ}(G) = 2 \cdot \mathrm{SJ}(F) = 2n$, so the estimator will be a factor of 2 off with probability at least p. ∎

2.4 Comparison of the algorithms

In both algorithms *sample-count* and *tug-of-war*, a single random variable is expected to provide the right estimate. However, in order to guarantee that for any input set, algorithm *sample-count* produces an accurate estimate with high probability, we need to have a sample of size $O(\sqrt{t})$. 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 algorithm *tug-of-war* is more demanding in its update time, which is proportional to the sample size. More importantly perhaps, the analysis given by [AMS96] provided theoretical bounds that apply in general to any input set. This leaves open the question as to which of the methods would demonstrate better performance in actual use. The experimental studies in the next section attempt to partially consider this issue.

3 Experimental Results

We have implemented the algorithms *sample-count*, *tug-of-war* and *naive-sampling*, and tested their performance on various data sequences. We used different data sets ranging from uniformly distributed random items to the sequences of words taken from the book *Wuthering Heights* and from *Genesis*. The data sets were either random according to some fixed distribution (like Poisson), excerpts from books,

Table 1: Data sets and their characteristics

data set	length	dom. size	self-join size	type	Fig.
path	40,800	40,001	680,000	artificial	1
zip1.5	120,000	2,184	$2.59398e+09$	statistical	2,3
zip1.0	500,000	9,994	$4.30435e+09$	statistical	4
uniform	1,000,000	32,768	$3.15176e+07$	statistical	5
mf2	19,998	1,693	$3.98391e+06$	statistical	6
mf3	19,968	2,881	618,664	statistical	7
selfsimilar	120,000	200	$3.40818e+09$	statistical	8
poisson	120,000	39	$9.11973e+08$	statistical	9
wuther	120,952	10,546	$1.11546e+08$	text	10
genesis	43,119	2,674	$2.30896e+07$	text	11
brown2	855,043	46,153	$5.83962e+09$	text	12
xout1	142,732	12,113	$9.17222e+07$	geometric	13
yout1	142,732	12,140	$9.45824e+07$	geometric	14

or geometric coordinates taken from spatial data. We also created an artificial data set designed to favor *tug-of-war* over *sample-count*.

Table 1 summarizes the data sets considered in this paper. For each data set, we list its length (n), its domain size (t), the actual self-join size, and its type, either artificial = artificially created, statistical = obtained using a statistical package, text = excerpts from well-known literary works, or geometric = coordinates taken from a spatial data set.

The performance was measured for sample sizes 2^i, for $i = 0, 1, 2, \ldots, 14$ (i.e., from 1 to 16,384). An example plot is given in Fig. 1. Plots for the other data sets appear in Figs. 3–14 at the end of the paper. In each plot, the labels on the x axis show the base two logarithm of the sample size. The labels on the y axis show 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 algorithms *sample-count*, *tug-of-war*, and *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 since each estimator is already based on the aggregation of many independent experiments.

3.1 Summary of the results

Algorithms *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. They appear to reliably estimate the self-join size of different kinds of sequences: both synthetic (from the Uniform, Zipf, Poisson, Self-similar, Multi-fractal distributions) and real (Wuthering Heights, Genesis, Brown Corpus, Spatial data).

In around half of the plots, 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.

The "path" data set was created in order to verify the theoretical analysis that there are data sets for which the *sample-count* algorithm converges particularly slowly (i.e., $\Theta(\sqrt{t})$ sample points are needed for an accurate estimate). The data set has 40,000 values that occur exactly once, and one value that occurs 800 times. The estimates for this pathological case are displayed in Figure 1, and indeed the performance closely matches the theoretical prediction.

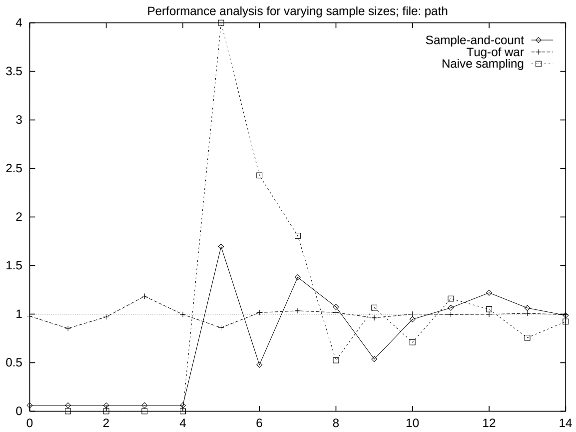

Figure 1: A pathological example, in which the three algorithms are run on a data set with 40,000 values occurring exactly once, and one value occurring 800 times. The x-axis depicts the base two logarithm of the sample size. The y-axis depicts the normalized value of the estimator, i.e., the ratio of the estimator to the actual self-join size. The horizontal line represents the target normalized value of 1. For each of the 3 algorithms, the normalized value of the estimator is plotted as a function of the sample size used to compute the estimator, for sample sizes 2^i, $i = 0, 1, 2, \ldots, 14$.

3.2 A closer look into the distribution 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 $X = X_{ij}$. Each such individual estimator X is the result of squaring the sum $Z = \sum_{j=1}^{n} h(v_j)$, for a single pseudo-random choice of a hash function $h : \{1, \ldots, t\} \mapsto$

$\{-1,1\}$.[2] In Fig. 2, we plot 103 individual estimators for a sequence generated according to the Zipf distribution with parameter 1.5. (The data set characteristics, including the actual self-join size, are given in the second row of Table 1.)

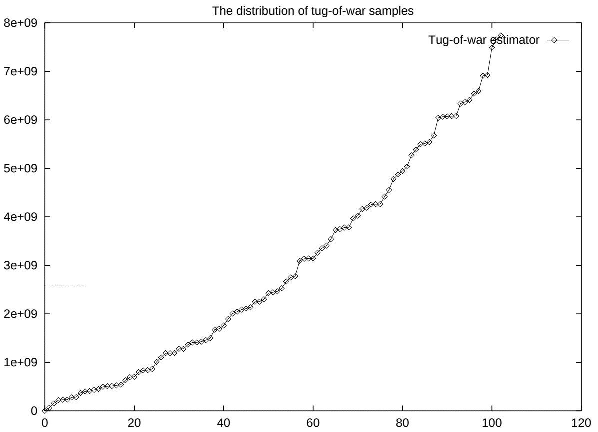

Figure 2: 103 different individual estimators X_{ij} produced by the *tug-of-war* algorithm run on data from the Zipf Distribution with parameter 1.5. The estimators have been sorted in increasing order. The value of the estimator is plotted as a function of the estimator number. Each estimator is based on a single sample point. The actual self-join size is depicted by a dashed horizontal line segment extending from the y-axis.

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, and ask if in this case, one can do better than random sampling?

[2] 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.

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. This algorithm is based on the tug-of-war approach outlined in the previous section. It also provides further motivation for tracking self-join sizes.

For simplicity, throughout this section we assume that all join sizes to be estimated are for pairwise equality joins on the same attribute. The results extend immediately to the case where the joins are on the same set of joining attributes. Extensions to handle the more general scenario of joins with different joining attributes are also straightforward, although typically additional space is required to keep track of the additional attributes.

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 $3|\Gamma_E|np^3 \leq |\Gamma_E|^2 p^4/\alpha$, 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, GMP97b], and hence one can track join sizes in limited storage 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) with 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, say, 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. We use a standard lower bound technique developed by Yao for a wide range of randomized models. 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 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 $\log n$ bits per relation. Using this signature as a building block, we construct a larger signature of $k \log n$ bits comprising k independent $\log n$ bit 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*

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

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

Proof.

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

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

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

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

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

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 performance 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 $k \log n$ 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 $2\cdot \mathrm{SJ}(F)\cdot \mathrm{SJ}(G) \le 2B_2^2$. Since the k-TW estimator is the arithmetic mean of k independent 1-TW estimator, we can upper bound its variance by $2\cdot \mathrm{SJ}(F)\cdot \mathrm{SJ}(G)/k \le 2B_2^2/k$. 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 $k \log n$ bits per 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.

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, implementation issues, and experimental evaluation. Extending our previous work [AMS96], 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 real and synthetic 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. Moreover, by a lower bound we show, 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. Moreover, the join signature for a relation can be maintained incrementally in the presence of insertions and deletions to the relation.

Future work includes performing an experimental study of the *tug-of-war* join signature scheme, 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.

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

References

[AGP99] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. Technical report, Bell Laboratories, Murray Hill, New Jersey, March 1999.

[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*, June 1999.

[AMS96] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. 28th ACM Symp. on the Theory of Computing*, pages 20–29, May 1996. Full version to appear in JCSS special issue for STOC'96.

[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.

[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.

[GM98a] 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.

[GM98b] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, 1998. To appear. Available as Bell Labs tech. rep., Sept. 1998, and at http://www.bell-labs.com/~pbgibbons/. Also, a two-page summary of this paper appeared in SODA'99.

[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.

[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*, 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.

[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.

[MRL98] G. S. Manku, S. Rajagopalan, and B. G. Lindsley. Approximate medians and other quantiles in one pass and with limited memory. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 426–435, June 1998.

[MS99] Y. Matias and E. Segal. Approximate iceberg queries. Technical report, Department of Computer Science, Tel Aviv University, Tel Aviv, Israel, March 1999.

[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.

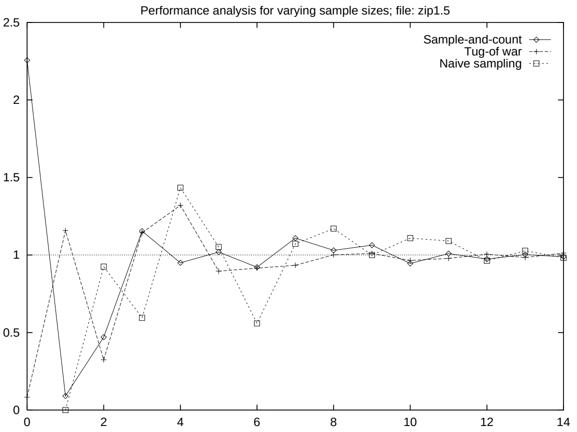

Figure 3: Accuracy comparison on data from the Zipf Distribution with parameter 1.5. The normalized value of the estimator produced by each of the 3 algorithms is plotted as a function of the base two logarithm of the sample size used.

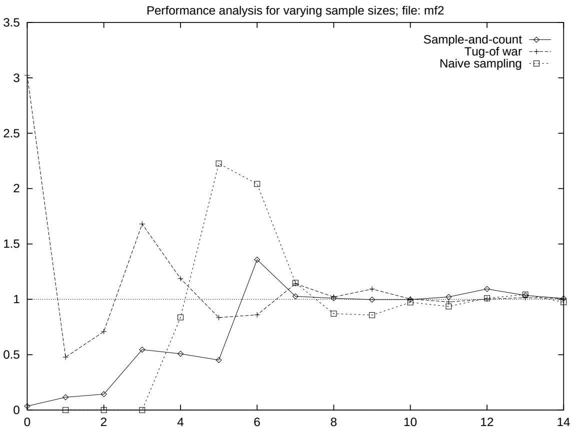

Figure 6: Accuracy comparison on data from the Multi-fractal distribution with parameters (20,000,0.2,12). The normalized value of the estimator produced by each algorithm is plotted as a function of the base two logarithm of the sample size used.

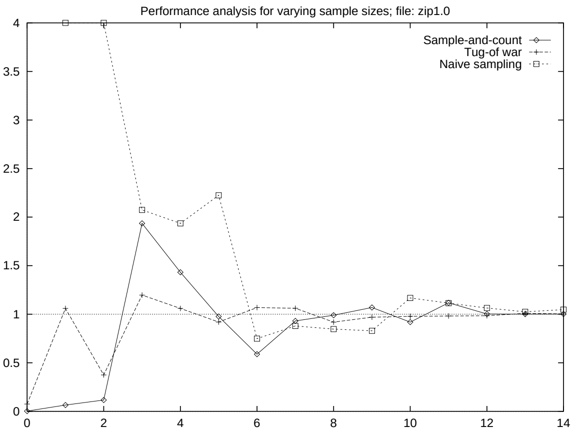

Figure 4: Accuracy comparison on data from the Zipf Distribution with parameter 1.0. The normalized value of the estimator produced by each algorithm is plotted as a function of the base two logarithm of the sample size used.

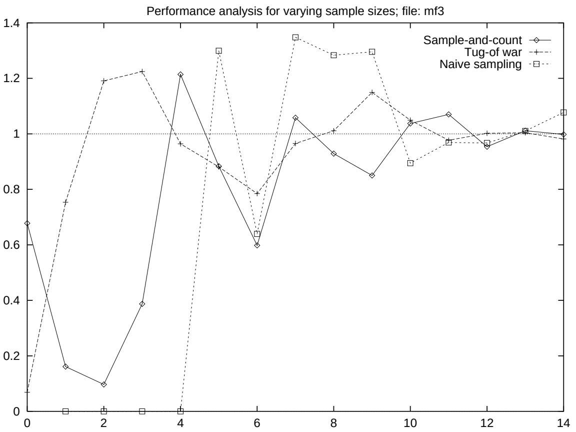

Figure 7: Accuracy comparison on data from the Multi-fractal distribution with parameters (20,000,0.3,12). The normalized value of the estimator produced by each algorithm is plotted as a function of the base two logarithm of the sample size used.

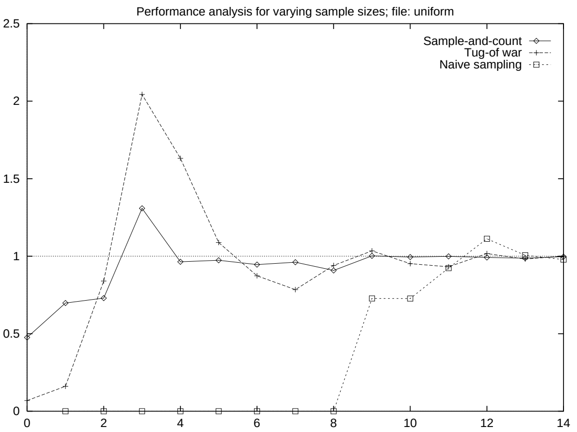

Figure 5: Accuracy comparison on data from the Uniform Distribution. The normalized value of the estimator produced by each algorithm is plotted as a function of the base two logarithm of the sample size used.

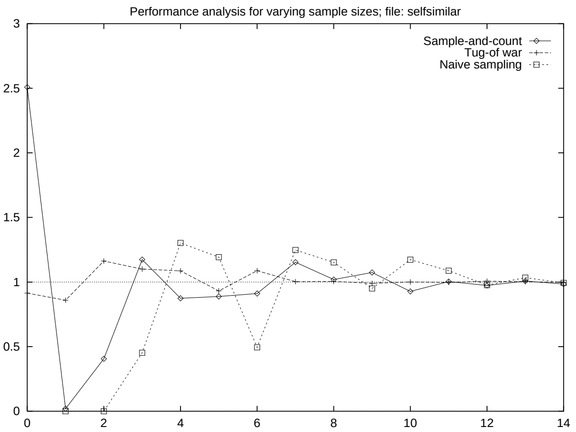

Figure 8: Accuracy comparison on data from the Selfsimilar Distribution. The normalized value of the estimator produced by each algorithm is plotted as a function of the base two logarithm of the sample size used.

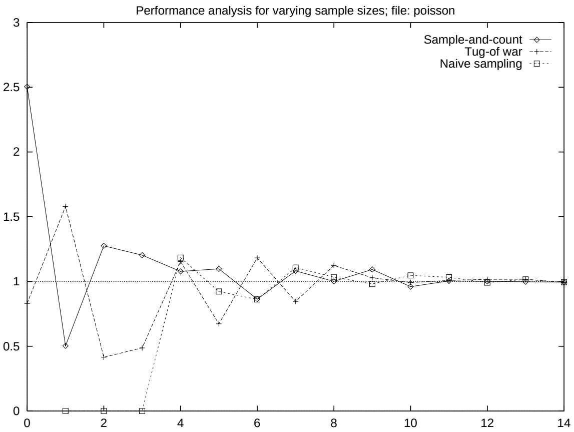

Figure 9: Accuracy comparison on data from the Poisson Distribution. The normalized value of the estimator produced by each algorithm is plotted as a function of the base two logarithm of the sample size used.

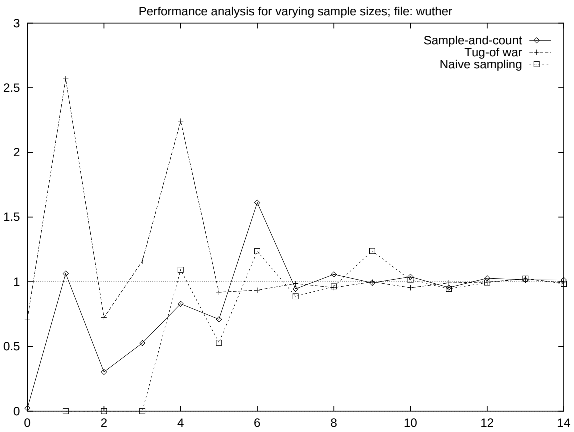

Figure 10: Accuracy comparison on words from the book Wuthering Heights. The normalized value of the estimator produced by each algorithm is plotted as a function of the base two logarithm of the sample size used.

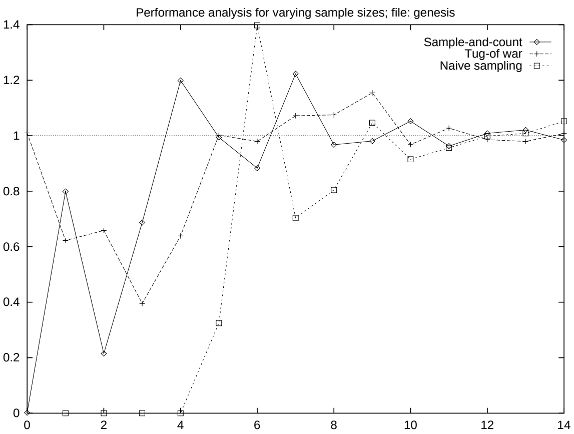

Figure 11: Accuracy comparison on words from the book of Genesis. The normalized value of the estimator produced by each algorithm is plotted as a function of the base two logarithm of the sample size used.

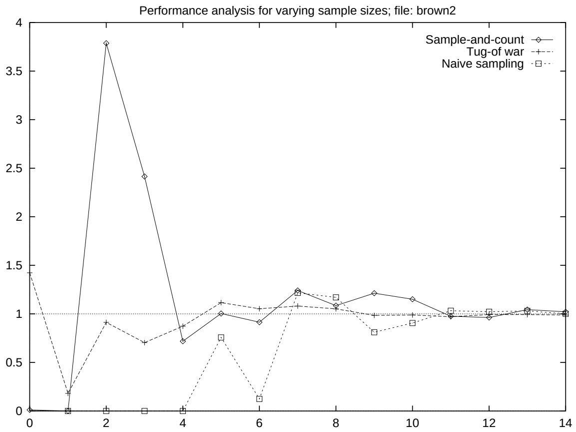

Figure 12: Accuracy comparison on words from the Brown Corpus. The normalized value of the estimator produced by each algorithm is plotted as a function of the base two logarithm of the sample size used.

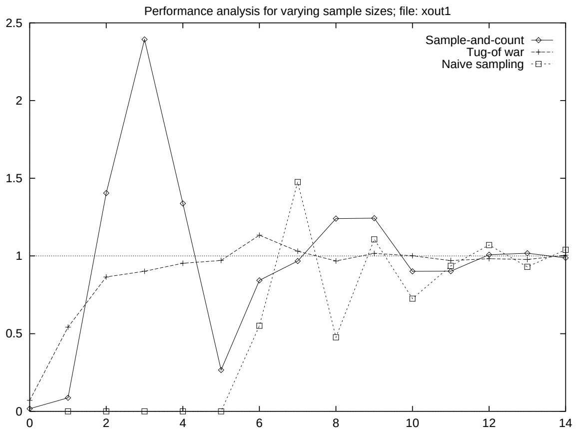

Figure 13: Accuracy comparison on the x-coordinates of data from a spatial point set. The normalized value of the estimator produced by each algorithm is plotted as a function of the base two logarithm of the sample size used.

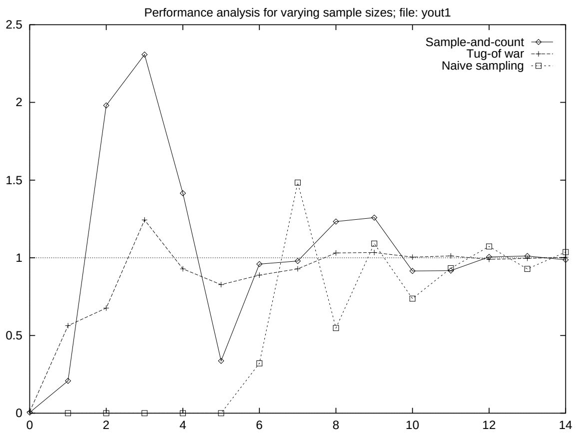

Figure 14: Accuracy comparison on the y-coordinates of data from a spatial point set. The normalized value of the estimator produced by each algorithm is plotted as a function of the base two logarithm of the sample size used.