Practical Selectivity Estimation through Adaptive Sampling

Richard J Lipton*
Department of Computer Science
Princeton University

Jeffrey F Naughton†
Department of Computer Sciences
University of Wisconsin

Donovan A Schneider‡
Department of Computer Sciences
University of Wisconsin

Abstract

Recently we have proposed an adaptive, random sampling algorithm for general query size estimation In earlier work we analyzed the asymptotic efficiency and accuracy of the algorithm, in this paper we investigate its practicality as applied to selects and joins First, we extend our previous analysis to provide significantly improved bounds on the amount of sampling necessary for a given level of accuracy Next, we provide "sanity bounds" to deal with queries for which the underlying data is extremely skewed or the query result is very small Finally, we report on the performance of the estimation algorithm as implemented in a host language on a commercial relational system The results are encouraging, even with this loose coupling between the estimation algorithm and the DBMS

1 Introduction

Estimates of query result size are useful in query optimization, as a means of determining the feasibility of queries, and as a quick way of answering queries for which the size of the answer is of interest in its own right The potential benefits of sampling-based algorithms for size estimation are many Unlike parametric methods [Chr83a, Chr83b, Dem80, Fed84, SAC+79, Lyn88, MDL83], they require no assumptions about the fit of the data to a probability distribution Unlike histogram or table based nonparametric methods [Chr83b, KK85, HTY82, Koo80, MD88, MK85, PSC84], they do not require storing and maintaining detailed statistics about the base data and views in the system Finally, they are robust in the presence of correlation of attributes, which allows accurate estimation for queries that involve many operators

However, there has been very little experimental or analytic work to evaluate the practicality of sampling estimation algorithms, perhaps due to skepticism about the performance of these algorithms Folk wisdom says that since

1 To attain reasonable accuracy, many samples must be taken, and

2 Sampling algorithms must do a disk I/O per tuple examined, whereas query evaluation algorithms can amortize the cost of a disk I/O over all tuples on a page, and

3 In sampling, the overhead of initiating an operation is incurred n times, where n is the number of samples taken, in evaluating the query, the overhead is incurred just once,

the cost of estimating a query through sampling is too high to be effective In this paper we argue that, to the contrary, a well-designed sampling algorithm for size estimation can be very efficient

Recently, we proposed the first adaptive random sampling algorithm for general query size estimation [LN89, LN90] The analysis in those papers showed that the algorithm has good asymptotic behavior, suggesting that it might be efficiently implementable However, asymptotics alone do not guarantee practicality The purpose of this paper is to

*Work supported by DARPA and ONR contracts N00014-85-C-0456 and N00014-85-K-0465, and by NSF Cooperative Agreement DCR-8420948

†Work supported by NSF grant IRI-8909795

‡Work supported by a DARPA/NASA Graduate Research Assistantship

demonstrate that the algorithm has sufficiently high performance to be useful in practice

We report the performance of the estimation algorithm in estimating the sizes of various select and join queries over a synthetic database designed to stress the algorithm To ensure that our tests did not underestimate the cost of sampling in a production quality database system, we implemented our algorithm as a host-level program running on a commercial relational database system (EQUEL and RTIngres) Note that this actually overestimates the cost of sampling, since the algorithm is implemented outside of the system, treating the system as a black box For example, in our implementation every sample requires a minimum of two UNIX pipe reads and two UNIX pipe writes as the host program communicates with the database back end

Detailed results of the experiments appear in Sections 4 and 5 The main point is that, unless the query itself can be computed extremely efficiently (e g , an equality selection on a key attribute with an index), or the answer is very small, the size of the query can be estimated accurately in a small fraction of the time it takes to compute the query

While implementing the sampling algorithm, several important points arose First, the algorithm gave much better estimates than were predicted by the bounds given in [LN89, LN90] In order to demonstrate that this is a property of the algorithm, and not of the specific data being used in the tests, we have done a new analysis of the algorithm in order to derive the smallest possible sampling bounds that guarantee the required confidence levels Section 2 provides this analysis The improvement over [LN89, LN90] is dramatic, for example, the bound for 95% confidence has been improved by a factor of 8 While this does not change the asymptotic time bounds, a factor of 8 reduction in running time can be the difference between a useful and a useless estimation algorithm

Second, highly skewed data provide special challenges to estimation through sampling To deal with this problem, in Section 3 we propose the notion of *sanity bounds* for sampling Intuitively, the adaptive algorithm augmented with sanity bounds will either (1) estimate the query size to within some given percentage of its true value, or (2) guarantee that the query size is itself small

The literature directly related to this paper is surprisingly sparse Both Piatetsky-Shapiro and Connell [PSC84] and Muralikrishna and DeWitt [MD88] discuss using sampling to build approximate selectivity histograms Those papers use the Kolmogorov test statistic to give bounds on the number of samples necessary to construct a histogram with a given accuracy, but do not consider the problem of estimating the size of the query directly through sampling

Olken and Rotem [OR86, OR89] consider the problem of sampling to construct a random subset of a query answer without computing the full answer This problem is complementary to size estimation, since an algorithm for size estimation does not imply an algorithm for constructing a random sample, and vice-versa

The most closely related work is that of Hou, Ozsoyoglu, and Taneja [HOT88, HOT89] In that work, the emphasis is on the estimation of aggregate queries in real-time environments, rather than on query size estimation The papers present data relating the number of samples to accuracy, but do not explicitly consider timing considerations A comparison in Sections 4 and 5 shows that while our algorithm and their algorithm are comparable for selections on single relations, for join queries our algorithm clearly dominates with respect to efficiency of size estimation

The results presented in this paper argue that size estimation through sampling could be easily added to database systems and can provide what is perhaps surprisingly good performance We close in Section 6 with a discussion that current trends in technology argue that sampling will be even more useful in future systems

2 Bounds on Sampling

The sampling algorithm presented in this paper is based on a model developed in [LN90] The central notion of that algorithm was that of *partitioning* the query In order to estimate the size of the query, we first partition the answer to the query into some number of disjoint subsets such that it is possible to randomly choose one of these subsets and compute its size We emphasize that this partitioning is conceptual, the sampling algorithm does not construct the answer to the query The sampling algorithm works by repeatedly randomly choosing one of these subsets, computing the size of the subset, then estimating the size of the query result based on these samples

Example 2.1 In this paper we will be concerned with the two "work-horse" operators of relational systems, join and select (The general algorithm applies to other types of queries as well)

First, consider a selection query $Q_1 = \sigma(R)$ In this case the answer can be partitioned based on the tuples in R Each tuple of R can be considered as a representative of a subset of the answer to the query, if the tuple satisfies the selection, then the size of the subset is 1, if not, the size of the subset is zero

Next, consider the natural join query $Q_2 = R \bowtie S$ The query is partionable as follows for each tuple r in R, the corresponding partition of Q_2 is all tuples t in Q_2 such that t was generated by joining r with some tuple of S In this case the size of a subset denoted by a tuple r is the number of S-tuples that join with r □

A novel feature of the estimation algorithm is that the termination condition is expressed in terms of the size of the sum of the samples taken, rather than in terms of the number of samples This lends the algorithm an adaptive flavor, if the samples are large, fewer will be taken, if the samples are small, more will be taken

If the size of a sample can be computed in time that is some function of the size of the sample, the adaptive nature of the algorithm makes it more efficient than a corresponding non-adaptive sampling approach Intuitively, this is because a non-adaptive sampling approach must take enough samples to guarantee accuracy in all cases, the adaptive algorithm is able to terminate early in the expensive cases, that is, when the samples turn out to be large

Suppose that the answer to the query to be estimated can be partitioned into n disjoint subsets, and define a random variable X to be the size of a randomly selected subset We let E denote the expected value of X, and V denote its variance

We assume that we have available two constants b and $A_{\max}$ These constants are specific to the query being estimated, b is an upper bound on the size of a partition, while $A_{\max}$ is an upper bound on the query size Note that $A_{\max}$ is just bn The accuracy does not depend on how close the bounds b and $A_{\max}$ are to their actual values, however, the closer they are, the more efficient the sampling

The sampling algorithm takes as parameters two integers d and e, and attempts to produce an estimate $\tilde{A}$ that is within $\max(A/d, A_{\max}/e)$ of the actual value A Additionally, a parameter p, where $0 \leq p < 1$, specifies the desired confidence in the estimate That is, the estimate will be within the specified error bound with probability p The general algorithm appears in Figure 1

The constants k_1 and k_2 depend on the desired confidence level p Initially, in [LN90], we presented the algorithm without the second conjunct in the control expression of the while loop (the conjunct $(m < k_2 e^2)$)) The reason for the second conjunct is given in Section 3 The analysis in that paper proved the following bound

Theorem 2.1 *Suppose that in a run of the algorithm of Figure 1, the while loop terminates because* $s \geq k_1 bd(d+1)$ *Then for* $0 < p \leq 1$, *if* $k_1 = 1/(1-\sqrt{p})$, *the error in* $\tilde{A}$ *is less than* A/d *with probability* p

```
s = 0,
m = 0,
while ((s < k1bd(d + 1)) and (m < k2e^2)) do begin
      s = s + RandomSample(),
      m = m + 1
end,
Ã = ns/m,
```

Figure 1 A general algorithm for query size estimation

Initial experiments with an implementation of the algorithm showed uniformly much better performance than that guaranteed by Theorem 2 1 The following theory offers a partial explanation of this phenomenon

Briefly, Theorem 2 1 gives weak bounds because it is so general In particular, it assumes that the sum of the samples has an arbitrary distribution, in practice, relatively few samples are necessary in order for the distribution of the sum to begin to look normal Note that this is not a statement about the distribution of the sizes of the partitions of the query Rather, it is an observation about the sum of a set of random samples of the partitions To quantify this observation, we use the following definition

Definition 2.1 Suppose that in a particular run of the algorithm in Figure 1 m samples are made Then we will say *the central limit approximation applies* if

$$\frac{\sum_{i=1}^{m} X_i - mE}{\sqrt{mV}}$$

has the standard normal distribution

By the Central Limit Theorem [Fel68], for any given instance of the estimation problem, for large enough m we may always treat the sampling as if the central limit approximation applies For small numbers of samples on certain distributions the approximation will be less accurate The following theorem shows that when the central limit approximation applies, much better bounds can be derived It uses the notation $\Phi(a) = 1/2\pi \int_{-\infty}^{a} e^{x^2/2} dx$, that is, the area under the unit normal distribution to the left of a

Theorem 2.2 *Suppose that in a run of the algorithm of Figure 1, the while loop terminates because* $s \geq$

p	k_1 (no CLA)	k_1 (with CLA)
0 80	9 5	2 6
0 90	19 5	3 8
0 95	39 5	5 0
0 99	199 5	12 2

Table 1 Value of k_1, with and without central limit assumption

$k_1 bd(d+1)$, and let the central limit approximation apply Then for $0 \leq p < 1$, if

$$k_1 \geq \left[\Phi^{-1}\left(\frac{1+\sqrt{p}}{2}\right)\right]^2$$

the error in $\tilde{A}$ is less than A/d with probability p

The improvement in the value of k_1 given by Theorem 2 2 over Theorem 2 1 is dramatic, the values of k_1 with and without the central limit assumption for several values of p are given in Table 1 (The reader uninterested in the proof of this theorem can safely skip the remainder of this section)

The proof of Theorem 2 2 uses the following sequence of lemmas We represent the probability of an event x by $P[x]$

Lemma 2.1 *Let $m = \beta V/E^2$ be a positive integer, and suppose that the sampling satisfies the central limit approximation Then*

$$P\left[\sum_{i=1}^{m} X_i \geq \frac{\alpha V}{E}\right] = 1 - \Phi\left(\frac{\alpha-\beta}{\sqrt{\beta}}\right)$$

Proof:

$$\begin{aligned} & P\left[\sum_{i=1}^m \tilde{X}_i \geq \alpha\frac{V}{E}\right] \\ = \; & P\left[\frac{\sum_{i=1}^m \tilde{X}_i - mE}{\sqrt{mV}} \geq \frac{\alpha V - mE^2}{E\sqrt{mV}}\right] \\ = \; & P\left[\frac{\sum_{i=1}^m \tilde{X}_i - mE}{\sqrt{mV}} \geq \frac{\alpha-\beta}{\sqrt{\beta}}\right] \\ = \; & 1 - \Phi\left(\frac{\alpha-\beta}{\sqrt{\beta}}\right) \end{aligned}$$

□

We bound the error in the estimate with the following lemma

Lemma 2.2 *Let $m \geq \beta V/E^2$ and $d > 0$, and suppose that the sampling satisfies the central limit approximation Then*

$$P\left[\left|\frac{n}{m}\left(\sum_{i=1}^{m}\tilde{X}_i\right) - A\right| \leq A/d\right] = 2\Phi\left(\frac{\sqrt{\beta}}{d}\right) - 1$$

Proof:

$$\begin{aligned} & P\left[\left|\tfrac{n}{m}\left(\textstyle\sum_{i=1}^m \tilde{X}_i\right) - A\right| \leq \tfrac{A}{d}\right] \\ = \; & P\left[\left|\frac{\sum_{i=1}^m X_i - mE}{\sqrt{mV}}\right| \leq \frac{mE}{d\sqrt{mV}}\right] \\ = \; & P\left[\left|\frac{\sum_{i=1}^m X_i - mE}{\sqrt{mV}}\right| \leq \frac{\sqrt{\beta}}{d}\right] \\ = \; & 2\Phi\left(\frac{\sqrt{\beta}}{d}\right) - 1 \end{aligned}$$

□

We can now prove Theorem 2 2

Proof: (Theorem 2 2) There are two parts to the proof First, we prove that if the algorithm in Figure 1 terminates with $s \geq k_1(V/E)d(d+1)$, then the desired confidence and accuracy hold Next, we show that $b \geq V/E$, completing the proof

There are two ways that the algorithm can fail — it can stop too early to guarantee a good error bound, or it can stop after enough samples but with a bad estimate

By Lemma 2 1, the probability that the algorithm stops with fewer than $m = \beta V/E^2$ samples is $\Phi((\alpha-\beta)/\sqrt{\beta})$ By Lemma 2 2, the probability that, given $m = \beta V/E^2$ samples, the estimate is within A/d of A, is $2\Phi(\sqrt{\beta}/d) - 1$ Combining the two gives a probability of success equal to

$$\Phi\left(\frac{\alpha-\beta}{\sqrt{\beta}}\right)\left(2\Phi\left(\frac{\sqrt{\beta}}{d}\right) - 1\right)$$

We can bound this product by setting the two probabilities to both be equal to p', where $p' = \sqrt{p}$, where p is the desired probability of success This gives $\beta = [\Phi^{-1}((1+p')/2)]^2 d^2$, and

$$\begin{aligned} \alpha = \; & \Phi^{-1}(p')\sqrt{\beta} + \beta \\ = \; & \Phi^{-1}\left(\tfrac{1+p'}{2}\right) d\left(\Phi^{-1}(p') + \Phi^{-1}\left(\tfrac{1+p'}{2}\right) d\right) \\ \leq \; & \Phi^{-1}\left(\tfrac{1+p'}{2}\right) d\left(\Phi^{-1}\left(\tfrac{1+p'}{2}\right) + \Phi^{-1}\left(\tfrac{1+p'}{2}\right) d\right) \\ \leq \; & \left[\Phi^{-1}\left(\tfrac{1+\sqrt{p}}{2}\right)\right]^2 d(d+1) \end{aligned}$$

This completes the proof that if the algorithm samples until $s \geq k_1(V/E)d(d+1)$, the desired accuracy holds with probability p

Now we turn to prove that $b \geq V/E$ By definition of V,

$$\begin{aligned} V \; &= \; \left(\frac{1}{n}\sum_1^n X_i^2 - \left(\frac{1}{n}\sum_1^n X_i\right)^2\right) \\ &\leq \; \frac{1}{n}\left(\sum_1^n X_i^2\right) \end{aligned}$$

$$\leq \frac{1}{n}\left(\sum_{1}^{n} X_i b\right)$$
$$\leq bE$$

so $V/E \leq bE/E = b$ This means that if $s \geq k_1 bd(d+1)$, we also have $s \geq k_1(V/E)d(d+1)$, and the theorem holds □

3 Skewed Data and Small Queries

While the central limit approximation indicates that in many cases a much smaller amount of sampling will suffice than the amount indicated by Theorem 2 1, there is still a problem of efficiency when b is large in comparison to E In practical terms, this means that the sizes of the partitions of the query are highly skewed, that is, that a large portion of the total query size is due to a small portion of the samples

The problem in this case is not so much with our specific algorithm, but with sampling in general To make the following discussion concrete, consider the case of estimating a selection on a 1,000,000 tuple relation, and, furthermore, that only one tuple satisfies the selection Then we will have 999,999 partitions of size zero, and one partition of size one This means that the expected size of a random sample is $1/1,000,000$, so sampling until $s > k_1 bd(d+1)$ can be expected to require $1,000,000 * k_1 d(d+1)$ samples

The problem is that the bound $s > k_1 bd(d+1)$ is designed to ensure that the total error is at most A/d In this case, that corresponds to asking for an error less than one on a sample space of size 1,000,000

The solution is to guarantee instead that the error will be at most some fixed fraction of the worst-case size In essence, if the answer is small relative to the problem space, we sample enough to guarantee that the answer is indeed small

As in the adaptive case, there are two types of bounds we can prove, depending on whether or not we assume the central limit approximation The following theorem does not assume the central limit approximation (For proofs of Theorems 3 1 and 3 2, see [LNS90])

Theorem 3 1 *Suppose that in a run of the algorithm of Figure 1, the while loop terminates because $m > k_2 e^2$ Then for $0 \leq p < 1$, if $k_2 \geq 1/(1-p)$, the error in $\tilde{A}$ is less than $A_{\max}/e$ of A with probability p*

If we assume that the central limit approximation is valid, tighter bounds are possible

p	k_2 (no CLA)	k_2 (with CLA)
0 80	5 0	1 7
0 90	10 0	2 7
0 95	20 0	3 8
0 99	100 0	6 7

Table 2 Value of k_2, with and without central limit assumption

Theorem 3.2 *Suppose that in a run of the algorithm of Figure 1, the while loop terminates because $m > k_2 e^2$, and suppose that the central limit approximation applies to the samples Then for $0 \leq p < 1$, if $k_2 \geq [\Phi^{-1}((1+p)/2)]^2$, the error in $\tilde{A}$ is less than $A_{\max}/e$ of A with probability p*

Again, as in the case of the bounds on k_1, the central limit approximation gives much better bounds The values for k_2 for several values of p are given in Table 2

When comparing these bounds with those given for those given in Section 2, it is important to note two things First, here the error is expressed in terms of the worst-case bound on the size of the actual sum being estimated, which in general may be much larger than the actual size of the sum That is, it is possible that $A_{\max}/d \gg A/d$ Second, here $k_2 e^2$ is the number of samples taken, whereas in the adaptive case $k_1 bd(d+1)$ is the sum of the sizes of the samples taken

Example 3 1 We return to the example of a selection query that returns a single tuple of a 1,000,000 tuple relation As noted above, if we omit the "sanity bound," and request that the query size be estimated to within 10% of the true query size with 80% certainty, we can expect to take about $5\ 0 \times 10^8$ samples Since for the same effort we could scan the entire relation 500 times, this is clearly a mistake

If we add the sanity bound and say that we wish to estimate the query to within 10% of the worst-case bound on the selection size or 10% of the answer size, whichever is larger, we get that at most 200 samples are required □

4 Estimating Selections

In the literature on estimating statistical parameters of database queries, estimating the fraction of tuples that satisfy a selection has received the most attention, both from a parametric and a non-parametric

viewpoint In this section we examine the problem of estimating the size of selection through sampling

Overview of Selectivity Sampling

Sampling to estimate selectivities is most effective in situations where the selection itself must be evaluated through a scan of the relation This includes selections on columns for which there is no index, complex selection conditions such as arithmetic expressions, and dyadic selection conditions such as "A X = A Y," where X and Y are attributes of relation A Sampling to estimate the selectivities of the last two types of selections is particularly important, since non-sampling parametric and non-parametric approaches do not work well in these cases

As noted in Example 2 1, our adaptive sampling algorithm can be used to estimate selectivities by partitioning the input relation by tuples That is, we randomly choose a tuple, and check to see if it satisfies the selection condition, if it does, the sample is of size one, if not, it is of size zero In order for this strategy to be practical, we need an index on some column of the relation, note that this need not be the column to which the selection applies

In the experiments we ran, we ensured that there was always an index on a dense key attribute of the relation (By "dense" we mean that there are no gaps between consecutive values appearing in the relation) This is not necessary in general, if we use the techniques from [OR89], all that is required is a B^+-tree on some attribute of the relation

If we assume that the cost of retrieving a sample tuple from the relation is much greater than the cost of verifying that the tuple satisfies the selection, then the form of the selection expression does not significantly affect the running time of the algorithm, all that matters is the selectivity being estimated Hence in our experiments we ran simple equality selections on columns without indices, but the results are also representative of estimating more complex selections with the same selectivities

The database used to estimate the selectivities was based on the database used in the Wisconsin benchmarks [BDT83] We tested 1% and 10% selectivities on relations of 10,000, 30,000 and 50,000 tuples Except where noted otherwise, the tuple size was 208 bytes We varied the selectivities by performing an equality select on a column with a random permutation of integers from one to ten (for the 10% case) or from one to 100 (for the 1% case)

The fragment of EQUEL code that does the sampling for the 1% case is presented in Figure 2 The column `a uniquela` is a key for relation `wiscrela`,

```
##  range of a is wiscrela
sum = m = 0,
while ((sum <= k*D*(D+1)*b) && (m <= k*E*E))
{
    m++,
    target = random() % N,
##  repeat retrieve (size = count(a uniquela
##     where a uniquela = @target and
##           a hundreda = 5))
    sum += size,
}
estimate = (sum*N)/m,
```

Figure 2 EQuel code for estimating selectivity

k	0 01 err	0 10 err	0 01 time	0 10 time
1 0	97	61	5 0	0 6
2 0	45	43	10 1	0 8
3 0	39	37	13 7	1 4
4 0	28	29	18 5	1 7

Table 3 Time and error vs k for selectivity tests

and there is a B^+-tree index on that column The relationship between this code fragment and the general algorithm in Figure 1 should be clear

Discussion of Selection Data

Table 3 gives the time to compute the estimate and the relative error in the estimate for both one and 10 percent selections from a 10,000 tuple relation and a range of k values Recall that the bound for the sampling is $k * d * (d+1) * b$, here b is 1, and in order to clarify the exposition, we also set $d = 1$ and vary k Since we wish to evaluate the adaptive nature of the algorithm, we set e artificially high, to avoid "sanity" escapes

The data points given represent average values over 100 trials All trials were run under RTIngres Release 5 0 running on a moderately loaded DEC VAX 6820

Two observations are clear from the data First, with the exception of the $k = 1\ 0$ value, the relative errors are very close for one and 10 percent Second, the one percent estimations take roughly a factor of ten longer to compute

These two observations are direct consequences of the adaptive nature of the sampling algorithm Intuitively, the algorithm samples until it has seen enough "one" samples to make a good estimate To see the

same number of "one" values in the one percent case as the 10 percent case, we would expect to do a factor of ten more samples, since there are a tenth as many ones

Note also that although the relative error is roughly the same in each case, the absolute error is a factor of ten worse in the 10 percent case For example, in the $k = 4\ 0$ entry, the error of 28% in the one percent selection corresponds to an error of 28 tuples, whereas the error of 29% in the ten percent selection corresponds to an error of 290 tuples This is the same observation that motivates the "sanity bounds" as discussed in Section 3

To put the efficiency of estimation into perspective, we compared the time of estimation to the time to actually compute the query Since both the one percent and 10 percent selections were chosen to force INGRES to scan the relation, both queries took the same time 9 0 seconds This means that while the 10 percent selection estimation was not too expensive, in all cases other than $k = 1\ 0$, estimating the size of the one percent selection took longer than computing the actual number

While on the surface this implies sampling is a bad idea for one percent selectivities, this is not necessarily the case First, the estimation ran for a long time because the sanity bounds were purposely set high enough to guarantee that they did not come into play In an actual application of sampling, the sanity bounds should be set so that the sampling will not run as long as the query

Secondly, and perhaps more importantly, sampling to estimate selectivities scales very well In fact, the time to estimate a selectivity is largely independent of the size of the input relation As the relation grows, the time to compute the query grows, so the percentage of compute time to estimate to a given accuracy decreases This is important because it is queries over large relations, where computation times can be very high, that must be estimated rather than computed The graph in Figure 3 gives data for estimating and computing a 10 percent selectivity over relations of 10K, 30K, and 50K tuples The vertical axis is the relative error of the estimate with respect to the actual query size, the horizontal axis is the ratio of the time to estimate the query size to the time to compute the query size

Another factor in the efficiency of sampling is the tuple size When scanning the relation to compute the answer, INGRES has the advantage of getting about c tuples for each disk access, where c is the average number of tuples per disk page Hence the larger c is, the less efficient sampling will be Conversely, with smaller c, sampling is more efficient

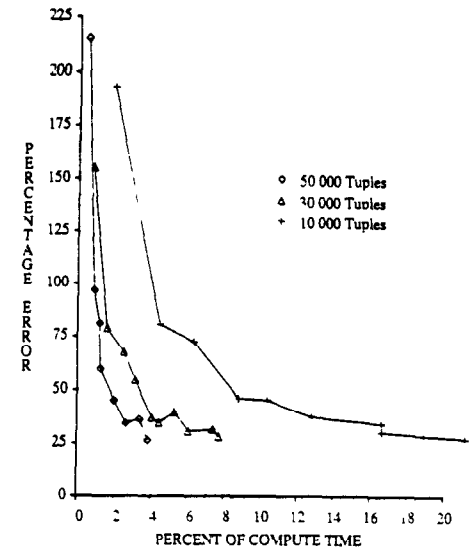

Figure 3 Scale-up results for select estimation

tup size	est	comp	est /comp
200 bytes	1 4	9 0	0 156
600 bytes	1 4	26 0	0 054

Table 4 Time vs tuple size for selectivity tests ($k = 3\ 0$, $s = 0\ 10$)

The simplest way to vary c is to vary the tuple size Table 4 shows the effect of varying the tuple size by a factor of three in the 10 percent selection The estimating time remains constant, whereas the time to compute the query grows in proportion to the tuple size The next subsection discusses two ways to deal with large c values

Comparison with Hou et al.

Hou, Ozsoyoglu, and Taneja [HOT88, HOT89] describe another algorithm for estimating selectivities through random sampling While similar, there are a number of differences between their approach and ours First, they sample without replacement, whereas we sample with replacement For the large population sizes we are considering, this does not make much of a difference Perhaps most significantly, Hou et al [HOT88] propose to use *cluster sampling* In cluster sampling, when a disk page is brought into memory, all tuples on the page are sampled This increases the efficiency, especially if c, the blocking factor, is large However, the samples are no longer independent

Using all tuples on a disk page is clearly a good idea One natural way to incorporate this into our sampling framework is as follows first, define a sample to be a randomly chosen disk page of the relation, rather than a randomly chosen tuple The size of a sample is the number of tuples in the page that satisfy the selection, and n, the number of partitions, is the number of disk blocks in the relation Note that this is different from cluster sampling, cluster sampling treats a disk page as a set of correlated samples, whereas the method outlined above treats a disk page as a single sample

From the EQUEL interface we could not implement either cluster sampling or this algorithm Intuitively, the block sampling algorithm of [HOT88] should perform better than the simple non-blocking adaptive sampling algorithm we have presented, since essentially in the same sampling time the blocking algorithm can examine a factor of c more tuples than the non-blocking adaptive algorithm The comparison between the algorithm of [HOT88] and the blocking version of our adaptive algorithm is less obvious, and will be the topic for future experimentation

5 Estimating Joins

In this section we consider the problem of estimating join selectivities through random sampling Estimating join selectivities is more involved than estimating the simple selectivities of Section 4, however, since in general computing a join is more expensive than computing a select, and much less is known about non-sampling methods of estimating joins, finding good sampling estimation algorithms for joins is critical

Overview of Join Sampling

In this paper we consider binary joins, although most of the discussion can be extended to arbitrary joins with minor modifications As discussed in Example 2 1, we sample joins by denoting one relation as the source relation and the other as the target relation The query answer is partitioned into as many subsets as there are tuples in the source relation, the size of a subset denoted by a tuple t of the source relation is just the number of tuples in the target relation that join with t A sample is just the size of a randomly chosen subset

The EQUEL fragment in Figure 4 implements the sampling loop for one of the joins described below Again, the analogy to the algorithm in Figure 1 should be clear

Because in join sampling the samples are no longer zero-one samples, the distribution of sample sizes can

```
## range of a is wiscrela
## range of d is wiscreld
sum = m = 0,
while ((sum <= k*b*D*(D+1)) && (m <= k*E*E))
{
    m++,
    target = random() % N,
##  repeat retrieve (size = count(d unique1d
##     where a unique1a = @target and
##           d norm8d = a thousa))
    sum += size;
}
estimate = (sum*N)/ m,
```

Figure 4 EQUEL to estimate a join size

be more interesting and has a significant effect on the estimation algorithm Recall from Section 2 that the bounds on sampling are expressed in terms of ratio of the variance and expected value of the samples, which is bounded above by the maximum size of any sample In order to fully test our algorithm, we investigated joins in which the join attribute in one relation was uniformly distributed, and the join attribute in the other was normally distributed By varying the standard deviation of the normal distributions, we were able to test the algorithm for various distributions of sample sizes

As in the selection estimation case, we assume that one of the relations has a dense key with an index, although all that is needed is a B^+-tree on some attribute There is an additional requirement for efficient join sampling after randomly choosing a tuple t of the source relation, we must find all tuples of the target relation that join with t Hence if the sampling is to be efficient, there must be an index on the join attribute of the target relation

In the following data, the source and target relations both contained 10,000 tuples, each of 208 bytes The relevant columns for the sampling were

- `unique1` — an integer key column, used for randomly choosing a source tuple
- `unique2` — a permutation of `unique1`
- `thousa` — random integers between one and 1000, subject to the condition that each appears exactly 10 times in the relation
- `normX` — the positive half of a normal distribution with mean zero and standard deviation X There are 3 such columns, for X = 250, 1000, and 16000

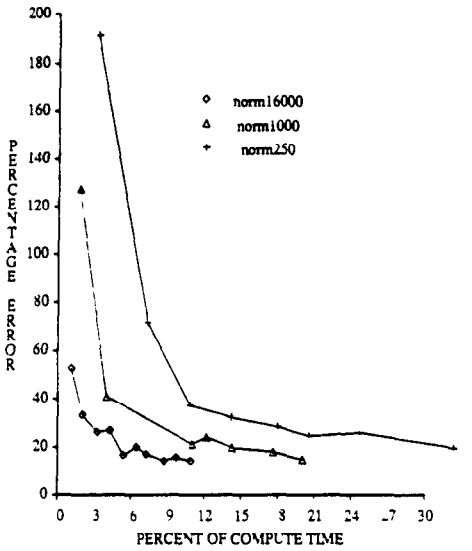

Figure 5 Error vs percentage of compute time

Discussion of Join Data

The graph in Figure 5 shows the relative error in the estimate as a function of the percentage of time to compute the full join There are curves for each of unique2 join normX for X = 250, 1000, and 16000 A couple of trends are clear from the graph First, fairly accurate estimates (certainly good enough to be useful in query optimization) are possible in a small fraction of the time required to compute the join size Second, the more skewed the data (the smaller the standard deviation) the less efficient the estimation

This can be understood as an extension of the situation with selects In the select case, if there were many zeroes and few ones, the algorithm had to sample for a relatively long time in order to discover those ones, in the join case, if there are a few outlying sample sizes that are much bigger than the average sample size, the algorithm again needs to do more sampling to discover those samples

This dependence on variance of the samples raises an interesting point there is no *a priori* reason to pick one relation as the source over the other Clearly, if only one relation has an index on the join attribute, it must be the target relation But if both relations have indices on the join attribute, either can serve as the source The decision should be made in such a way as to minimize the variance of the samples

Consider joining two relations, with the join attribute in one relation **thous**, and the join attribute in the other **norm2000** It turns out that if we make the relation with **norm2000** the source relation, then

time (sec)	err, $V/E = 1\ 14$	err, $V/E = 3\ 2$
0 25	20	29
0 50	14	20
0 75	12	17
1 00	10	15

Table 5 Two ways to sample a join

$V/E = 3\ 2$, whereas if we pick the relation with **thous** as the source, $V/E = 1\ 14$ Table 5 compares the performance of both strategies by giving the relative error of each strategy at several points in time The data points are averages over 100 trials, the errors for each time t are interpolated values The strategy with lower V/E converges faster

The scale-up considerations in terms of tuple size for join estimation are similar to those for select estimation However, if V/E grows as the size of the joining relations grows, then since the amount of sampling necessary for a given accuracy is proportional to V/E, the cost of sampling will not remain constant On the other hand, if V/E grows more slowly than the size of the relation, or remains constant, as is often the case, join sampling will increase in efficiency as relations grow

Comparison with Hou et al.

Hou et al [HOT88, HOT89] also give an algorithm for estimating join selectivities In its simplest form, the algorithm works as follows Suppose we are joining relations R and S, and denote the size of R by $|R|$, and the size of S by $|S|$ The algorithm works as follows First pick a tuple from the R, then pick a tuple from S If the tuples join, the sample is of size one, otherwise, the sample is of size zero Say m such samples have been taken, and that the sum is s Then estimate that A, the true answer, is $|R||S|s/m$

A more sophisticated version of their algorithm, also presented in [HOT88], involves randomly choosing a set of disk blocks of R and a set of disk blocks of S, then comparing all tuples within these blocks to see if they join This greatly reduces the number of disk I/O's for a given number of tuple comparisons

Again, since we cannot implement this blocked sampling algorithm through the EQUEL interface, we cannot sensibly compare times for the two algorithms However, we can compare the accuracy of the estimate in terms of the number of tuples examined

We compared relative accuracies of our adaptive sampling algorithm and the HOT algorithm for the query **unique2a** join **quarterb** That is, the join at-

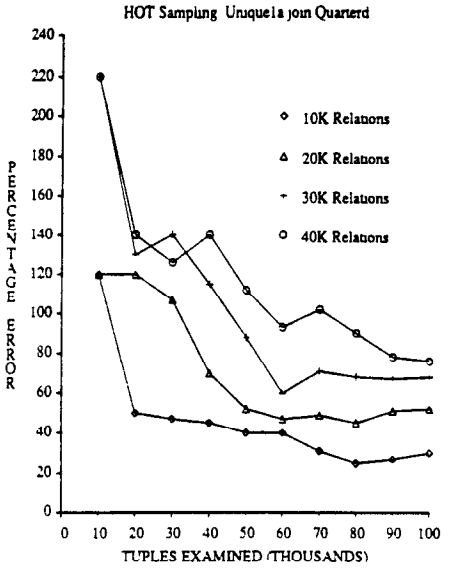

Figure 6 Accuracy of HOT Algorithm for query `unique2a` join `quarterb`

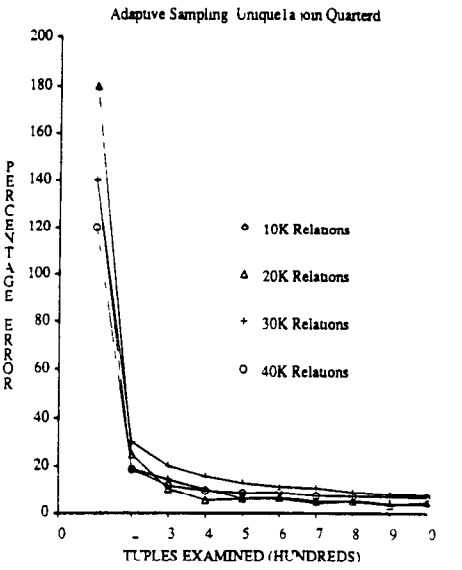

Figure 7 Accuracy of Adaptive Algorithm for query `unique2a` join `quarterb`

tribute in relation A contained a random permutation of the integers from zero to n, where n is the number of tuples in relation A, while the join attribute in relation B contained a random permutation of the integers from zero to $n/4$, each integer appearing four times

Figure 6 gives the accuracy of the HOT algorithm on this query for relations of sizes varying from 10K to 40K tuples Figure 7 gives the accuracy of the adaptive algorithm for the same query Both graphs compare the relative error in the estimate vs the number of tuples examined The scale for the x-axis is different in the two figures, the graphs indicate that the adaptive algorithm converges to a good estimate much faster (in terms of number of tuple comparisons) than the HOT algorithm However, since the HOT algorithm uses clustered sampling, it is able to make more tuple comparisons per disk I/O than the adaptive algorithm

Also, the intended application of the HOT algorithm is real-time systems, where predictability of the time to compute a sample is paramount Since every sample in the HOT algorithm consists of extracting a fixed number of disk pages, it is predictable, whereas in the adaptive algorithm, the time for a sample will vary depending on the size of the sample

6 Conclusion

We have argued that adaptive random sampling can be a useful tool in estimating query sizes Implemented in a loosely coupled manner as a host language program, it gave good performance over a wide range of select and join queries

In future work we intend to examine the efficiency of random adaptive sampling when added as an operator within the database system This will allow us to test the disk block-at-a-time variant of our select estimation algorithm, and to test the performance of the algorithm with the host/system overhead removed Finally, we plan to investigate sampling on large main-memory and on multiprocessor machines Such machines provide an extremely attractive environment for size estimation through sampling, since 1) in main-memory databases, there is no handicap to the sampling algorithm due to poor use of disk accesses, and 2) on a multiprocessor, many samples can be done simultaneously in parallel

References

[BDT83] D Bitton, D DeWitt, and C Turbyfill Benchmarking database systems A systematic approach In *Proc Ninth VLDB*, pages 8–19, 1983

[Chr83a] S Christodoulakis Estimating block transfers and join sizes In *Proc ACM*

SIGMOD Conference, pages 40–54, May 1983

[Chr83b] S Christodoulakis Estimating record selectivities *Information Systems*, 8(2) 69–79, 1983

[Dem80] R Demolombe Estimation of the number of tuples satisfying a query expressed in predicate calculus language In *Proc Sixth VLDB*, pages 55–63, 1980

[Fed84] J Fedorowicz Database performance evaluation using multiple regression techniques In *Proc ACM SIGMOD Conference*, pages 70–76, June 1984

[Fel68] W Feller *An Introduction to Probability Theory and Its Applications*, volume 1 John Wiley and Sons, Inc , New York, New York, 1968

[HOT88] W -C Hou, G Ozsoyoglu, and B Taneja Statistical estimators for relational algebra expressions In *Proc ACM PODS*, pages 276–287, March 1988

[HOT89] W -C Hou, G Ozsoyoglu, and B Taneja Processing aggregate relational queries with hard time constraints In *Proc ACM SIGMOD Conference*, pages 68–77, June 1989

[HTY82] L Herschberg, P D Ting, and S B Yao Query optimization in star computer networks *ACM Transactions on Database Systems*, 7(4), December 1982

[KK85] N Kamel and R King A model of data distribution based on texture analysis In *Proc ACM SIGMOD Conference*, pages 319–325, May 1985

[Koo80] R Kooi *The optimization of queries in relational database systems* PhD thesis, Case Western University, Cleveland, Ohio, 1980

[LN89] R Lipton and J Naughton Estimating the size of generalized transitive closures In *Proc Fifteenth VLDB*, pages 165–172, August 1989

[LN90] R Lipton and J Naughton Query size estimation by adaptive sampling In *Proc ACM PODS*, March 1990

[LNS90] R Lipton and J Naughton and D Schneider Practical Selectivity Estimation through Adaptive Sampling University of Wisconsin-Madison Computer Sciences Department Technical Report, March 1990

[Lyn88] C Lynch Selectivity estimation and query optimization in large databases with highly skewed distributions of column values In *Proc Fourteenth VLDB*, pages 240–251, August 1988

[MCS88] M Mannino, P Chu, and T Sager Statistical profile estimation in database systems *Computing Surveys*, 20(3) 191–221, September 1988

[MD88] M Muralikrishna and D DeWitt Equi-depth histograms for estimating selectivity factors for multi-dimensional queries In *Proc SIGMOD Conference*, pages 28–36, June 1988

[MDL83] A Montgomery, D D'Souza, and S Lee The cost of relational algebraic operations on skewed data Estimates and experiments *Information Processing Letters*, pages 235–241, 1983

[MK85] B Muthuswamy and G Kerschberg A DDSM for relational query optimization Technical report, University of South Carolina, Columbia, 1985 As cited in [MCS88]

[OR86] F Olken and D Rotem Simple random sampling for relational databases In *Proc Twelfth VLDB*, pages 160–169, August 1986

[OR89] F Olken and D Rotem Random sampling from B$^+$trees In *Proc Fifteenth VLDB*, pages 269–278, August 1989

[PSC84] G Piatetsky-Shapiro and C Connell Accurate estimation of the number of tuples satisfying a condition In *Proc ACM SIGMOD Conference*, pages 256–276, June 1984

[SAC+79] P G Selinger, M M Astrahan, D D Chamberlin, R A Lorie, and T G Price Access path selection in a relational database management system In *Proc ACM SIGMOD Conference*, pages 23–34, 1979