ON ESTIMATING THE SIZE OF PROJECTIONS

by

Jeffrey F. Naughton and S. Seshadri

Computer Sciences Technical Report #1021

April 1991

On Estimating the Size of Projections*

Jeffrey F. Naughton and S. Seshadri
Computer Sciences Department
University of Wisconsin-Madison

Abstract

We present a new sampling algorithm for estimating the number of tuples in the projection of a relation. The algorithm requires no assumptions about the distributions of values in the attributes of the relation and converges faster and smoother than previous sampling algorithms for the problem. We give both a sound theoretical basis for the algorithm and experimental data from an implementation of the algorithm.

1 Introduction

We present a new sampling algorithm for estimating the number of tuples in the projection of a relation onto a subset of its attributes. Unlike previous parametric techniques, our algorithm requires no assumptions about the distributions of values in the attributes and is robust in the presence of correlations between the attributes. Over a wide range of data, our algorithm converges faster than previous sampling algorithms, and furthermore the convergence is smoother and is easier to predict.

Estimating the size of the result of a relational query is an integral part of cost evaluation. Cost evaluation in turn is useful both in query optimization and in system resource allocation. Another application of cost evaluation that is becoming increasingly important is online information retrieval. Currently, in such systems a user submits a query, and receives in return both the answer to the query and a bill for the resources used during query evaluation. It would be preferable to provide the user with an estimate of the cost before the query is run, and to give the user the option of withdrawing the query based upon that estimate.

The algorithm we present for estimating the size of a project operation is based upon the adaptive random sampling framework that we developed in [LN90]. The framework is described in more detail in Section 3; here we note that in order to use the framework, one must

*Work supported by NSF grant IRI-8909795

- Choose a conceptual partitioning of the answer to the query into disjoint subsets, and
- Devise an algorithm for randomly choosing a subset of the query result and computing its size.

The estimating algorithm proceeds by repeatedly choosing a random subset of the query result and computing its size until it has gained enough information to guarantee a good estimate for the size of the entire answer to the query.

Designing a good partitioning strategy and an efficient algorithm for computing a sample is the critical factor in the performance of an estimation algorithm derived from this framework. For the project operator, this problem is particularly challenging.

In [LN89] we considered the problem of estimating the size of the transitive closure of a relation, while in [LNS90], we considered the problem of estimating the sizes of the answers to selects and joins. For those operations, there is a clear one-to-one correspondence between tuples of the input relations and certain disjoint subsets of the output relation. In that case, the partitions of the result can be these disjoint subsets, and a sample can be computed by randomly choosing an input tuple t and computing the size of the subset of the answer that t determines. By contrast, under a projection, many tuples of the input relation can map to the same tuple of the output relation, so this straightforward approach will not work.

Our partitioning strategy involves conceptually partitioning the query answer into subsets at a finer granularity than a single tuple. That is, it is possible that a tuple of the answer relation is viewed as being split into some number of partitions. Our sampling strategy makes use of an index on one of the target attributes of the project. The use of the index is designed so that the algorithm can compute a sample while examining only the relevant portion of the input relation. This property of our sampling strategy, coupled with memoing to avoid redundant effort over multiple samples, allows the size of the project to be estimated with a small fraction of the effort required to compute the projection.

The remainder of this extended abstract is organized as follows. Section 2 discusses previous work on estimating the size of projections. Section 3 reviews the adaptive sampling framework developed in [LN90, LNS90]. Section 4 presents our sampling algorithm, which we call "index project sampling." Section 5 gives experimental data for a series of experiments we ran on an implementation of our estimating algorithm. Section 6 contains conclusions and problems for future research.

2 Related Work

Estimating the size of a projection is an intriguing problem, and is one that has received a fair amount of attention in the literature. In most previous work on the problem,

researchers have assumed some distribution of the input relation and, based on that assumption, derived a formula for the size of the projection.

Gelenbe and Gardy [GG82a] assume that the attributes of the relation are uniformly and independently distributed over fixed finite domains, and then use combinatorial arguments to derive formulas for the probability that the projection is of a given size. In [GG82b] the authors extend this work to consider functional dependencies, and derive formulas for the expected size of the projection (rather than the probabilities for each possible size.)

If the assumptions on which the formulas are based are indeed valid, the formulas give exact answers. However, any deviation from the assumptions can produce large errors. For example, if two attributes over the same domain are strongly correlated instead of being independent, the estimate may be off by a factor proportional to the size of the domain. Furthermore, there is no way to guarantee the quality of the estimate, since it depends on untested assumptions. By contrast, our sampling algorithm requires no assumptions about distributions or correlations, and produces results guaranteed to be within a user-specified accuracy with a user-specified confidence.

Gardy and Puech [GP84] present an approach based on generating functions. Rather than assuming uniformly, independently distributed attributes, this paper assumes that for each tuple t in the domain of the database, the probability that t appears in the input relation is known. Again, if this information is indeed available, the estimate will be exact. However, as with the uniform assumption, the error in the estimate could be large if the information is only approximate.

Merrett and Otoo [MO79] propose a model in which the relation is considered as a set of points in a multidimensional space, with one dimension for each attribute. They partition each dimension into sectors, and assume that each attribute is uniformly distributed within each sector, and that the attributes are independently distributed. The formula for the project size requires constants obtained via a scan of the input relation, so the time for a scan is a lower bound on the time for the estimate. Ahad, Bapa Rao, and McLeod [ABM89] show that given some semantic constraints on the database, a similar formula can be derived without a scan. Again, both methods suffer large errors if the underlying assumptions of uniformity and independence are not valid.

Whang, Vander-Zanden, and Taylor [WVZT90] give an algorithm for probabilistic counting that can be used to estimate the size of a projection in linear time and in a small, constant fraction of the storage required to hold the input relation. The algorithm presented by Flajolet and Martin [FM85] probabilistically estimates the size of a projection in linear time and space logarithmic in the size of the input relation. The time required for a scan is a lower bound on the time for the estimate for both of the above algorithms too.

Hou, Ozsoyoglu, and Taneja in [HOT88, HOT89] note that a statistical method known as Goodman's estimator [Goo49] can be used to estimate the size of projections. As is

discussed in Section 5, this technique does not require an index and has an interesting behavior: up until a certain threshold number of samples (the threshold is different for each query), it produces wildly inaccurate estimates. After the threshold, the estimator converges very quickly. In Section 5, we show that in general our algorithm produces better estimates below the threshold, and comparable estimates above the threshold.

3 Adaptive Random Sampling

The sampling algorithm presented in this paper is based on a framework developed in [LN90] and [LNS90]. To make this paper self-contained, in this section we repeat the key results of those papers.

The central notion of that framework is that of *partitioning* the answer to 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. 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.

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.

In the following we let A be the quantity we wish to estimate, that is, it is the number of tuples in the answer to a given query. Suppose that the answer to the query 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 also assume that we have available the constant $A_{\max}$, an upper bound on the query answer size.

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 algorithm presented in Figure 1 is a general algorithm for size estimation. In that algorithm, the constants k_1 and k_2 depend on the desired confidence level p.

The accuracy of Algorithm 3.1 is given by the following theorem:

Theorem 3.1 *Suppose that in a run of Algorithm 3.1, the while loop terminates because* $s \geq k_1 d(d+1)V/E$. *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.

Since V/E will not be known, an upper bound on V/E is used to compute the stopping conditions for the algorithm. This is considered in more detail in Section 4.

Algorithm 3.1 (Size Estimation)
$s := 0$;
$m := 0$;
while $((s < k_1 d(d+1)V/E)$ and $(m < k_2 e^2))$ do begin
    $s := s +$ *RandomSample()*;
    $m := m + 1$
end;
$\check{A} := ns/m$;

Figure 1: A general algorithm for size estimation.

As presented above, the algorithm is inefficient when b is large in comparison to E. In that case, the number of samples required to reach the adaptive termination condition $s \geq k_1 d(d+1)V/E$ will be large. To avoid oversampling in such a situation, we give a "sanity bound" that provides a stopping conditionsbased on the number of samples rather than the sum of the samples.

Theorem 3.2 *Suppose that in a run of Algorithm 3.1, 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* $\check{A}$ *is less than* $A_{\max}/e$ *with probability* p.

4 Index Project Sampling

To apply the framework developed in Section 3 to estimating the number of tuples in the result of project operations, we need to decide what a sample is and how a sample can be computed. As mentioned in the introduction, the problem is complicated by the fact that many tuples of the input relation can produce the same output tuple.

A first attempt at a sampling strategy is as follows: let R be the input relation, and suppose we wish to estimate $|\pi(R)|$, where $\pi(R)$ represents the projection of R onto a subset of its attributes. Furthermore, let R' represent R after the unwanted attributes have been deleted from each tuple of R, but before duplicates have been eliminated.

Consider some tuple $t \in R$. Let t' be the projection of t onto the target attributes of the projection being computed. Then if t' appears d times in R', we consider t as contributing $1/d$ to the size of the projection $\pi(R)$. Another way to view this is that we partition $\pi(R)$ at a granularity finer than a tuple — that is, each tuple $t' \in \pi(R)$ is split into d partitions, where d is the number of times that the tuple t' appears in R'.

Under this scheme, a sample can be computed by randomly choosing some tuple t from R, and computing how many times its projection t' appears in R'. Choosing a random tuple t is not hard. In the following, we will assume that there is an index on a

dense key attribute of R, so we can select a random tuple by generating a random key value and using the index to get the corresponding tuple. If we have no index on a dense key of R, all we require is a B^+-tree (using the algorithm of Olken and Rotem [OR89]) or a hash table (using the algorithm of Olken, Rotem, and Xu [ORX90]) on some attribute of R.

Computing the number of times the projection t' of a randomly chosen tuple t appears in R' is more problematic. For our solution we require an index on some attribute of R that appears in $\pi(R)$. This index can be used to select a subset S_t of R such that all tuples in R whose projection is t' appear in S_t. This can be accomplished as follows.

Suppose that R has the schema $R(K, A, B)$, where attribute K is a key for R, and the project we wish to estimate is $\pi_{A,B}(R)$. (Here A and B need not be single attributes; in general, they can be sets of attributes. Furthermore, if the key K is a compound key, both A and B can contain subsets of K.) Furthermore, suppose we have indexes on $R.K$ and $R.A$. Then we sample as follows:

1) choose a random key k;
2) lookup, using the index on $R.K$, the tuple t with key k;
3) lookup, using the index on $R.A$, the set of tuples $S_t = \{t_i | t_i.A = t.A\}$;
4) generate S'_t by deleting all attributes but A and B from S_t;
5) let $\pi(t) = t'$ appear d times in S'_t;
5) return $1/d$;

This scheme is correct, but a closer examination reveals that it is wasteful in the following sense: after the lookup at line 3, the set S_t will in general have many tuples t_i, all of which have $t_i.A = t.A$ but only some of which have $t_i.B = t.B$. The tuples such that $t_i.A = t.A$ but $t_i.B \neq t.B$ are ignored.

This can be remedied as follows: define π_i to be $|\pi_{A,B}(S_t)|$, and l_i to be $|S_t|$. Then for any tuple s in S_t, we define the contribution of s to the projection to be π_i/l_i. This corresponds to partitioning $\pi(R)$ in the following manner:

1. First, group $\pi(R)$ by $R.A$ value.
2. Consider one such group, with $R.A$ value a. If π_i is the number of tuples in the group, and l_i is the number of tuples in R with $R.A$ value a in R, then partition the group into l_i equal subparts.

Note that in general π_i/l_i is not an integer, so again we have partitioned $\pi(R)$ into subsets at a granularity finer than individual tuples. We illustrate the definition and computation of a sample with the following example.

Example 4.1 Let the relation R have schema $R(K, A, B)$ and consist of the tuples

$$(1, a_1, b_1)$$

$(2, a_1, b_1)$
$(3, a_1, b_1)$
$(4, a_1, b_2)$
$(5, a_2, b_1)$

Then if the randomly chosen key k is one of 1, 2, 3, or 4, the set S_t contains the tuples $(1, a_1, b_1)$, $(2, a_1, b_1)$, $(3, a_1, b_1)$, and $(4, a_1, b_2)$. The projection $\pi_{A,B}(S_t)$ contains the tuples (a_1, b_1), and (a_1, b_2). Here $\pi_i = 2$, and $l_i = 4$, so the sample returns 0.5.

If the randomly chosen key k is 5, the set S_t contains the single tuple $(5, a_2, b_1)$. The projection $\pi_{A,B}(S_t)$ contains the tuple (a_2, b_1), so here $\pi_i = 1$, and $l_i = 1$, and the sample returns 1.0.

The answer to the project is

(a_1, b_1)
(a_1, b_2)
(a_2, b_1)

Grouping the answer by $R.A$ value, we find there are two groups which are partitioned into 5 disjoint "subsets" as follows: (a_1, b_1) and (a_1, b_2) are together split into four subsets, while (a_2, b_1) forms a subset by itself. □

Figure 2 presents the Index Project algorithm.

In function ProjSample, the first time a given $t.A$ value is seen, the corresponding sample value is stored in a table. Subsequently, if another tuple with the same $t.A$ value is seen, the stored result from the table is returned immediately instead of being recomputed. To bound the accuracy of the estimate produced, we use the following proposition.

Proposition 4.1 *Let X_i be a random variable denoting the value of a sample as defined by procedure ProjSample(). Then $V[X_i]/E[X_i] \leq 1$.*

This proposition, together with Theorem 3.1 from Section 3 and the fact that for project operations, if the input size is n, then $A_{\max} = n$, we have the following theorems:

Theorem 4.1 *Suppose that in a run of Algorithm 4.1, the while loop terminates because $s > k_1 d(d+1)$ Also, let the size of the input relation be n. Then for $0 \leq p < 1$, if $k_1 = 1/(1-\sqrt{p})$, the error in $\tilde{A}$ is less than n/d with probability p.*

Theorem 4.2 *Suppose that in a run of Algorithm 4.1, the while loop terminates because $m > k_2 e^2$ Also, let the size of the input relation be n. Then for $0 \leq p < 1$, if $k_2 \geq 1/(1-p)$, the error in $\tilde{A}$ is less than n/e with probability p.*

Algorithm 4.1 (Index Project)

```
float function ProjSample();
begin
1)    choose random key k;
2)    lookup, using index on R.K, the tuple t such that t.K = k;
3)    if (InTable(t.A)) then
4)          return Table[t.A]
5)    else
6)          lookup, using index on R.A, the set S_t = {t_i | t_i.A = t.A};
7)          compute l_i = |S_t|;
8)          compute π_i,the number of distinct answer tuples in S_t;
9)          Table[t.A] := π_i/l_i;
10)         return π_i/l_i;
11)   end; /* else */
end; /* ProjSample */

float function IndexProject();
begin
1)    s := 0.0;
2)    m := 0.0;
3)    while ((s < k_1 d(d+1)) and (m < k_2 e^2)) do begin
4)          s := s + ProjSample();
5)          m := m + 1
6)    end;
7)    return ns/m; /* n = |R| */
end; /* IndexProject */
```

Figure 2: The Index Project algorithm.

5 Performance

To investigate the performance of our estimation algorithm, we implemented the algorithm and ran a series of experiments. The implementation consisted of the C code necessary to implement the sampling algorithm, together with the code required to implement in-memory relations and their associated indexes.

Throughout, the metric we used was the number of tuples examined. We chose this metric because it is independent of parameters of the implementation, yet still highly indicative of performance. Since we wished to investigate the relationship between the number of tuples examined and the error in the estimate, we ignored the stopping conditions and let the algorithm perform a large number of samples, recording the current error after each sample. In all cases the data points are averaged over 100 trials.

Also, all tests were run relations with schema $R(K, A, B)$. The attributes K, A, and B were all integer valued. In every case, K was a key for R, while A and B were varied to stress the estimating algorithm in different ways. We built indexes on $R.K$ and $R.A$.

The first point to notice is that the index lookup at line 6) of function ProjSample() can be inefficient if the indexed attribute has a large number of duplicates. For example, in the extreme case, if the same value a appears in $R.A$ in every tuple of R, then the lookup will return the entire relation. We regard such a situation as unlikely, since an index on such an attribute would be of very little use for any purpose. However, to investigate the dependence of our estimation algorithm on the distribution of this attribute we ran the following series of tests for three values of the relation R.

In each case, the relation R had 10000 tuples, which were divided into two parts, each of 5000 tuples. In the first part, all tuples had the same B value, and there were 1250 distinct A values, uniformly distributed. In the second part, there were 5000 distinct B values. The three instances of R correspond to the following three distributions of A values in the tuples of the second part of R:

A1: Five distinct values, grouped in five bins of size 1000.

A2: 50 distinct values, grouped in 50 bins of size 100.

A3: 5000 distinct values, grouped in 5000 bins of size one.

These relations are designed to keep constant the number of tuples in the input, number of tuples in the output, and the ratio V/E of the samples. This isolates the effect of varying the A distribution.

A graph of error versus number of tuples examined for each of the three cases appears in Figure 3. The dashed line in that graph corresponds to data for Goodman's estimator, which is discussed at the end of this section.

As expected, the more uniform the A distribution, the better the convergence of the algorithm. In the following, we will always use "reasonable" distributions for the A attribute, that is, no more than 5% of the A values are identical.

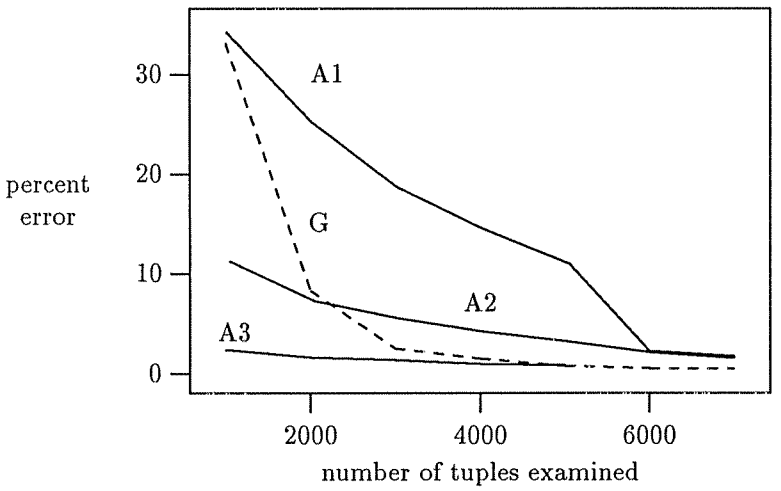

Figure 3: Dependence on R.A distribution for index project sampling.

The second point of interest is the dependence of the estimating algorithm on the "projectivity" of the projection, that is, the ratio of the answer size to the input size. We ran tests on three instances of R, each instance containing 10,000 tuples. In each case, the attribute $R.A$ consisted of 10,000 values distributed as the positive half of the normal distribution about zero with variance 1000. We tested two distributions of $R.B$:

P1: Two distinct values, uniformly distributed. This relation had a result size of 3740 tuples, for a projectivity of 0.37.

P2: 10 distinct values, uniformly distributed. This relation had a result size of 7723 tuples, for a projectivity of 0.77.

A graph of the results are given in Figure 4. The graph demonstrates the general trend that the higher the projectivity (the larger the result), the faster the convergence.

The third point of interest is the comparison between our sampling algorithm with the one based upon Goodman's estimator [Goo49]. Goodman's estimator works as follows: first, select m tuples at random from R, where R has n tuples. Next, delete unwanted attributes from the sampled tuples, and group the resulting tuples by the number of duplicates. That is, group i contains all tuples that, after deleting the projected-out attributes, appear i times in the sample. Let the number of tuples in group i be x_i. The Goodman's estimator estimates that the project has $\sum_{i>0} A_i x_i$ tuples, where

$$A_i = 1 - (-1)^i \frac{(n-m+i-1)^{\underline{i}}}{m^{\underline{i}}}$$

with $n^{\underline{i}} = n(n-1)\ldots(n-i+1)$ if $i > 0$, and $n^{\underline{0}} = 1$.

An obvious advantage of using Goodman's estimator is that it does not require an index on any of the project attributes. This also implies that Goodman's estimator

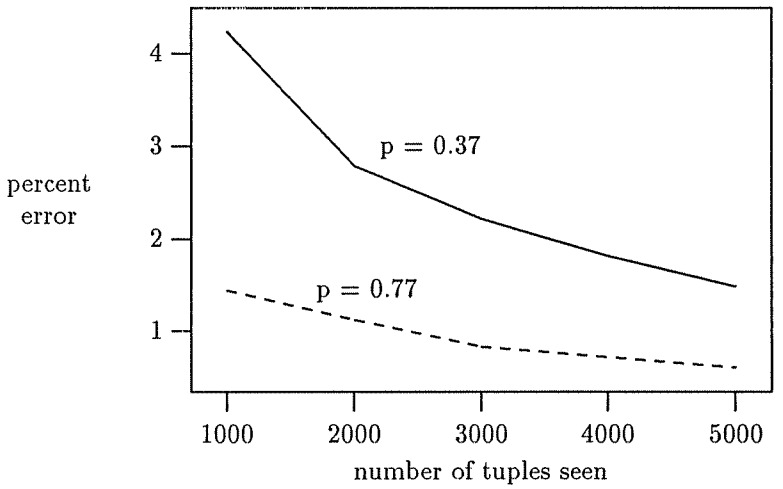

Figure 4: Dependence on projectivity for index project sampling.

depends only on the distribution of the duplicates, hence is insensitive to the distribution of the attribute $R.A$. For this reason, Goodman's estimator outperformed index select for the most skewed $R.A$ distribution curve in Figure 3 and also for large sample sizes for the second most skewed $R.A$ distribution curve of the graph. (The performance of Goodman's estimator is shown in the dashed line of that graph.)

The behavior of Goodman's estimator on the relations tested in Figure 4 is graphed in Figure 5. Note that the vertical axis in Figure 4 is expanded by a factor of more than 50 inFigure 5. Intuitively, for small numbers of samples, the absolute value of A_i in Goodman's estimator are very large, and any random variance of the distribution of duplicates seen in the sampling is multiplied tremendously. Since the A_i also grow with i, this effect is more pronounced at lower projectivities.

Notice that the estimator can produce very high or negative estimates when the absolute value of the A_i are very large. As noted by Goodman, his estimator can be modified to give reasonable estimates by noticing that an upper bound on the answer is n and a lower bound is the number of distinct answer tuples we have seen in the samples. This eliminates "unreasonable" estimates, but does not speed the convergence to good estimates. This also makes the estimator biased. For these reasons, our comparisons are with the unmodified estimator.

An important property of an estimation algorithm is how it scales as the input relation size grows. That is, we wish to answer the question "if the size of the input relation doubles, what happens to the number of tuples that must be examined to achieve a given accuracy?" The final set of experiments were designed to compare index project with Goodman's estimator under this metric.

We scaled the relation size in two ways. In the first, we keep the ratio of the project size to the input size constant. We refer to this sort of scaleup as "constant ratio scaleup."

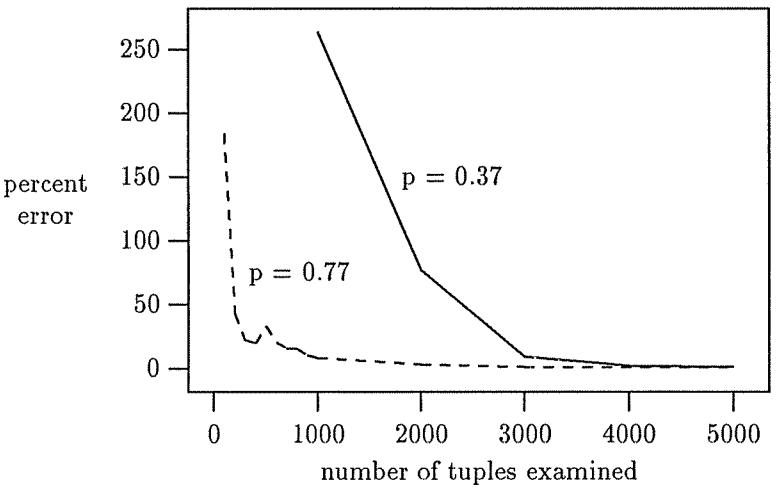

Figure 5: Dependence on projectivity for Goodman's estimator.

In the second, we keep the size of the result constant while increasing the size of the input relation. We refer to this sort of scaleup as "constant project scaleup."

Figure 6 gives a graph of number of tuples examined vs. relation size for the constant ratio scaleup for the index project algorithm (solid line) and for Goodman's estimator (dashed line.) The relations used in that graph contained the following distribution of duplicates (after deleting the attribute $R.K$ from each tuple):

- $1000k$ tuples appeared once,
- $1000k$ tuples appeared twice,
- $1000k$ tuples appeared three times,
- $1000k$ tuples appeared four times,

for $k = 1, 2, 5$, and 10. The graph plots the number of tuples examined to achieve 10% error vs. relation size. In the graph, the line for the index project algorithm remains constant, although the scale is too large to show it.

Figure 7 gives the corresponding graph for the constant project scaleup for the index project algorithm (solid line) and for Goodman's estimator (dashed line.) There the relations used relations with the following distribution of duplicates (again, after deleting the attribute $R.K$ from each tuple):

- 1000 tuples appeared k times,
- 1000 tuples appeared $2k$ times,
- 1000 tuples appeared $3k$ times, and
- 1000 tuples appeared $4k$ times,

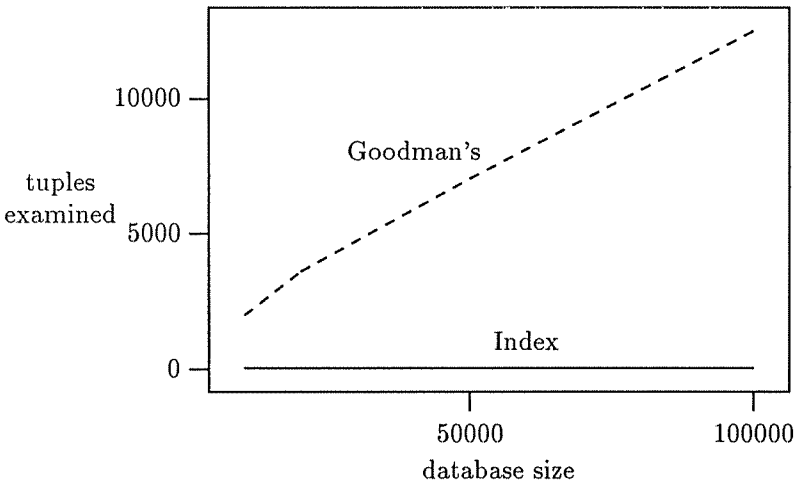

Figure 6: Scaleup for 10% error, constant ratio scaleup.

again for $k = 1, 2, 5$ and 10. The graph plots the number of tuples examined to achieve 10% error vs. relation size. In the graph, the line for the index project algorithm grows linearly, although again the scale is too large to show it.

It is clear that in both cases, index project scaled much better than Goodman's estimator. The performance of index project is especially good under constant ratio scaleup, where the number of tuples required for a given accuracy is essentially independent of relation size.

6 Conclusion

We have presented a new sampling algorithm for estimating the number of tuples in the result of a projection on a relation. Experimental evidence verifies that if there is an index on one of the projected-upon attributes, this sampling algorithm can be used to produce accurate estimates while examining only a small subset of the tuples of the input relation.

This sampling algorithm, which we call index project, has the advantage over parametric methods in that its accuracy is guaranteed without imposing restrictions on the distributions of attributes of the relation. Also, when compared with Goodman's estimator, the index project algorithm produces much better estimates below the critical number of samples for Goodman's estimator, and comparable estimates above that threshold number of samples. Finally, the index project algorithm scales well as the relation size increases.

These results indicate that the index project algorithm is a useful tool for estimating the sizes of projects, hence will be useful in query cost estimation. We are currently investigating an extension of the project estimation algorithm that uses multi-level sampling

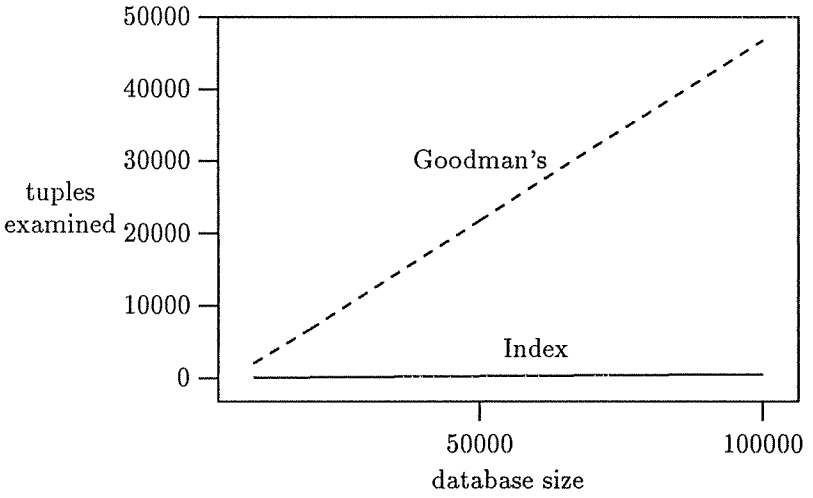

Figure 7: Scaleup for 10% error, constant project scaleup.

to eliminate the need for an index, and are also investigating techniques for incorporating the index project algorithm with our previous algorithms for estimating the sizes of selects and joins.

References

[ABM89] Rafiul Ahad, K. V. Bapa Rao, and Dennis McLeod. On estimating the cardinality of the projection of a database relation. *ACM Transactions on Database Systems*, 14(1):28–40, March 1989.

[FM85] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *JCSS*, 31, 1985.

[GG82a] Erol Gelenbe and Daniele Gardy. On the sizes of projections: I. *Information Processing Letters*, 14(1):18–21, March 1982.

[GG82b] Erol Gelenbe and Daniele Gardy. On the sizes of projections: II. In *Proceedings of the 8th VLDB Conference*, Mexico City, Mexico, September 1982.

[Goo49] L. A. Goodman. On the estimation of the number of classes in a population. *Ann. Math. Sta.*, 1949.

[GP84] Daniele Gardy and Claude Puech. On the sizes of projections: A generating function approach. *Information Systems*, 9(3/4):231–235, 1984.

[HOT88] Wen-Chi Hou, Gultekin Ozsoyoglu, and Baldeao K. Taneja. Statistical estimators for relational algebra expressions. In *Proceedings of the Seventh ACM*

Symposium on Principles of Database Systems, pages 276–287, Austin, Texas, March 1988.

[HOT89] Wen-Chi Hou, Gultekin Ozsoyoglu, and Baldeao K. Taneja. Processing aggregate relational queries with hard time constraints. In *Proceedings of the ACM-SIGMOD Conference on the Management of Data*, pages 68–77, Portland, Oregon, June 1989.

[LN89] Richard J. Lipton and Jeffrey F. Naughton. Estimating the size of generalized transitive closures. In *Proceedings of the Fifteenth International Conference on Very Large Databases*, pages 165–172, Amsterdam, The Netherlands, August 1989.

[LN90] Richard J. Lipton and Jeffrey F. Naughton. Query size estimation by adaptive sampling. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Nashville, Tennessee, March 1990.

[LNS90] Richard A. Lipton, Jeffrey F. Naughton, and Donovan A. Schneider. Practical selectivity estimation through adaptive sampling. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Atlantic City, New Jersey, May 1990.

[MO79] T. H. Merrett and E. Otoo. Distribution models of relations. In *Proceedings of the 5th International VLDB Conference*, pages 418–425, Rio de Janeiro, Brazil, October 1979.

[OR89] Frank Olken and Doron Rotem. Random sampling from B^+-trees. In *Proceedings of the Fifteenth International Conference on Very Large Databases*, pages 269–278, Amsterdam, The Netherlands, August 1989.

[ORX90] Frank Olken, Doron Rotem, and Ping Xu. Random sampling from hash files. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 375–386, Atlantic City, New Jersey, May 1990.

[WVZT90] Kyu-Young Whang, Brad T. Vander-Zanden, and Howard M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems*, 15(2):208–229, June 1990.