A Linear-Time Probabilistic Counting Algorithm for Database Applications

KYU-YOUNG WHANG
Korea Advanced Institute of Science and Technology
BRAD T. VANDER-ZANDEN
Cornell University
and
HOWARD M. TAYLOR
University of Delaware

We present a probabilistic algorithm for counting the number of unique values in the presence of duplicates. This algorithm has $O(q)$ time complexity, where q is the number of values including duplicates, and produces an estimation with an arbitrary accuracy prespecified by the user using only a small amount of space. Traditionally, accurate counts of unique values were obtained by sorting, which has $O(q \log q)$ time complexity. Our technique, called *linear counting*, is based on hashing. We present a comprehensive theoretical and experimental analysis of linear counting. The analysis reveals an interesting result: A load factor (number of unique values/hash table size) much larger than 1.0 (e.g., 12) can be used for accurate estimation (e.g., 1% of error). We present this technique with two important applications to database problems: namely, (1) obtaining the column cardinality (the number of unique values in a column of a relation) and (2) obtaining the join selectivity (the number of unique values in the join column resulting from an unconditional join divided by the number of unique join column values in the relation to be joined). These two parameters are important statistics that are used in relational query optimization and physical database design.

Categories and Subject Descriptors: G.3 **[Mathematics of Computing]**: Probability and Statistics—*probabilistic algorithms*; G.4 **[Mathematics of Computing]**: Mathematical Software—*algorithm analysis*; H.2.2 **[Database Management]**: Physical Design; H.2.4 **[Database Management]**: Systems—*query processing*

General Terms: Algorithms, Experimentation, Performance, Theory

Additional Key Words and Phrases: Bit map, column cardinality, hashing, join selectivity, physical database design, query optimization, statistical databases

1. INTRODUCTION

The number of unique values in a column (*column cardinality*) of a relation is one of the most important statistics that is used in query optimization and physical database design. Many database models and commercial systems use

Authors' addresses: K.-Y. Whang, Computer Science Department, Korea Advanced Institute of Science and Technology, P.O. Box 150, Cheong-Ryang ui, Seoul, Korea; B. T. Vander-Zanden, Computer Science Department, Cornell University, Ithaca, NY 14853; H. M. Taylor, Department of Mathematical Sciences, University of Delaware, Newark, DE 19716.

this statistic to estimate the selectivity of a predicate, the size of the answer set (final or intermediate) of a query, or the size of a projection [4, 6, 11, 18]. In query optimization, the estimated values are used to select the minimum-cost access plan for executing a query [11]. In physical database design, they are used to find the access configuration that gives the minimum average response time for a set of user transactions [15]. Thus, accurate estimation of the column cardinality is crucial in obtaining good database performance.

Unfortunately, the presence of duplicates in a column makes it difficult to maintain the column cardinality dynamically at every insertion or deletion of a tuple. Maintaining column indexes could alleviate the problem, but in many practical applications, indexes are not maintained for all columns because of maintenance overhead.

Conventional techniques that have been proposed to estimate the column cardinality can be classified into separate categories—sorting, hashing, and sampling. Sorting schemes require $O(q \log q)$ disk accesses [2] and therefore are inefficient. Bitton and DeWitt [2] propose a scheme that removes duplicates during the sorting process. They present an example in which the number of disk accesses is reduced by as much as 50 percent. Nonetheless, the method still requires $O(q \log q)$ disk accesses.

Hashing represents a second approach to determining the column cardinality. Hashing has the nice property of eliminating duplicates without a need to sort. Thus, only one scan of the relation is required. If the key values are stored in the hash table, an exact count of unique values in a column can be obtained. However, simple application of hashing could do worse than sorting because, except for relations with relatively low column cardinalities, the hash table would be too large to fit in main memory and thus would have to be stored on disk. Each probe into the hash table would then cost approximately one disk access and the counting algorithm would require $(q + q/k + q/b)$ disk accesses, where k is the average number of tuples having the same column value (*duplication factor*), and b is the number of tuples stored in each disk block (*blocking factor*). The first term in the expression counts the number of disk accesses for probing (reading) the hash table, the second for writing new hash table entries (for unique values) and the third for scanning the relation. Note that this number is quite possibly larger than the $2 \times (q/b + (q/b)\log_z(q/b))$ disk accesses required by the sorting algorithm using an external z-way sort merge [17] in practical ranges of b, q, and z. For example, for $b = 100$, $q = 1$ million, $z = 10$, sorting would cause 100,000 disk accesses, while simple hashing would cause 1.11 million disk accesses, assuming a duplication factor of 10.

Finally, random sampling would intuitively appear to offer an attractive mechanism for estimating the column cardinality. However, taking a simple random sample from the relation causes the proportion of distinct values to the relation cardinality to be overrepresented because, in a small sample, the probability of hitting the same value twice is fairly small. Thus, simple linear extrapolation will overestimate the number of distinct values in the column significantly. For example, when we take a random sample of one percent of the relation, we cannot simply multiply the number of distinct values in the sample by one hundred to get the estimated number of distinct values in the full set. In general, to get the proper estimate, we must use precise distribution information.

However, this information is more complicated to obtain than the column cardinality. The same observation is made in [10] in a slightly different context, where the number of disk accesses is counted under random sampling. Olken and Rotem [9] make the same observation by pointing out that projection and sampling cannot be commuted. Hence, projection after sampling would produce a different number of distinct values than sampling after projection would. The main reason for this phenomenon is the presence of duplicate values. Note that counting the number of unique values is essentially a projection operation, in which duplicates are eliminated. Therefore, we cannot obtain a valid estimate by sampling before counting.

In this paper we present a probabilistic technique that estimates the number of unique values in a column or a group of columns in linear time—$(c \times q/b)$, where q is the number of tuples in the relation (*relation cardinality*), b is the blocking factor, and c is either one or two depending on whether the *relation cardinality* (number of tuples in a relation) is available. If the relation cardinality is not known, a preliminary scan of the relation must be made to determine the cardinality. Our technique is based on hashing. However, unlike the simple application of hashing, the algorithm we propose does not store the key values in the hash table; instead, it simply turns on a bit in the appropriate entry in the hash table. Since two records that share the same hash table entry (i.e., collide) cannot be distinguished, an exact count can no longer be obtained. Thus, the column cardinality must be approximated based on the occupancy rate of the hash table when the relation has been completely scanned. In this sense the algorithm is probabilistic.

We demonstrate that our algorithm can obtain the column cardinality for fairly large relations with an arbitrarily small error, using a bit map[1] that can fit in main memory. For example, with 1.25 Mbytes of main memory available for the map, the algorithm can be used for relations as large as 120 million tuples with an error of 1 percent. The only penalty is the loss of EXACT COUNT. The algorithm also provides significant enhancement in processing time compared with conventional techniques. For instance, it would cause only 1,000 disk accesses for the example we discussed above.

Compared with simple hashing, the algorithm significantly reduces the storage space needed for the hash table for two reasons. First, by storing bits rather than keys and by avoiding overflow records, it reduces the size of the hash table entry. For example, for four-byte integer keys, it reduces the space by a factor of 32. For 20-character string keys, this factor increases to 160. Second, a surprising result of the analysis in Section 4 is that *load factors* (the number of unique values/map size) much higher than one (e.g., twelve) can be used while achieving good accuracy (e.g., 1%). Accordingly, the hash table size can be further reduced by a factor of 12. Note that in simple hashing the load factor cannot exceed one. Overall, the algorithm can obtain storage reduction by a factor of 384 for integer keys and 1,920 for 20-character string keys.

A sketch of this algorithm, termed *linear counting*, was presented in [1] without a detailed analysis, along with two other algorithms also based on hashing:

[1] A *map* is a structure that keeps track of the statistical behavior of the counting process. In linear counting the map is a hash table.

logarithmic counting and *sample counting*. In linear counting, the estimator is based on a measurement that is approximately proportional to (linear in) the true count. Hence, the name linear counting. In logarithmic counting and sample counting, the estimators are based on measurements that are logarithms of the true counts.

It was pointed out in [1] that linear counting provides the most accurate estimates of any of the three algorithms. Further, linear counting can be made arbitrarily accurate since the technique allows the programmer to allocate increasingly large maps without paying a performance penalty (one probe for each element in the column). Increasing the size of the map degrades the performance of the other methods since they require as many probes as the map size for each element.

Despite the accuracy of the linear counting method, Astrahan et al. suggested that linear counting could not be advantageously used for relations whose cardinality exceeded 20 million, since the map size would become burdensomely large. This limitation was deemed necessary since it was believed that the load factor had to be kept below two to maintain a reasonable accuracy over a large range of column cardinalities. Thus, a relation with 20 million tuples would require a 10 million-entry map. Since each entry of the map occupies one bit, the map requires 1.25 Mbytes of storage space. Any map larger than this would probably have to be stored on disk, and the resulting I/O accesses would cause a significant degradation of performance.

As previously mentioned, our analysis discloses that load factors much higher than two (e.g., twelve) can be used while achieving good accuracy. Since a large load factor reduces the map size, our analysis suggests that linear counting can be profitably used for much larger relations than previously thought. This aspect is discussed in detail together with interesting constraints determining the map size.

We proceed by first reviewing the basic counting algorithm described in [1]. We then present a detailed analysis and develop a complete algorithm for practical applications. This analysis focuses on two properties of the ratio $\hat{n}/n$, where $\hat{n}$ denotes the maximum likelihood estimator of the true count of unique values (i.e., column cardinality) n:

(1) the bias in expected value and
(2) the standard error.

The *bias* of the ratio $\hat{n}/n$ is equal to the expected relative error of the estimator $\hat{n}$. For example, a bias of .01 indicates that, on average, the estimator $\hat{n}$ will overestimate the column cardinality n by 1 percent. Formally, the bias is defined as in Eq. (1).

$$\text{Bias}(\hat{n}/n) = E(\hat{n}/n) - 1. \tag{1}$$

The *standard error* is defined as the standard deviation of $\hat{n}/n$. It indicates how variable the ratio $\hat{n}/n$ is expected to be, indicating the extent of the error in the measurement.

The rest of the paper is organized as follows. Section 2 sketches the basic linear counting algorithm. Section 3 presents our analysis of the bias and the

standard error of the measurement. Section 4 introduces constraints that apply to the counting process to achieve the desired standard error and to avoid invalid measurements. We provide techniques to satisfy these constraints. In Section 5 we present the results of comprehensive experiments performed to verify the analysis. Section 6 presents a complete linear counting algorithm that can be readily applied to practical applications. In Section 7 we discuss an interesting application of linear counting that obtains *join selectivity*, that is, the number of unique values in the join column resulting from an unconditional join divided by the number of unique join column values in the relation to be joined. An unconditional join is a join without any selection predicates. The join selectivity is an important parameter in determining the size of the answer set of a query [4]. Finally, we conclude the paper in Section 8.

2. LINEAR COUNTING: THE BASIC ALGORITHM

Linear counting is a two-step process. In step 1, the algorithm allocates a *bit map* (hash table) of size m in main memory. All entries are initialized to "0"s. The algorithm then scans the relation and applies a hash function to each data value in the column of interest. The hash function generates a bit map address and the algorithm sets this addressed bit to "1". In step 2, the algorithm first counts the number of empty bit map entries (equivalently, the number of "0" entries). It then estimates the column cardinality by dividing this count by the bit map size m (thus obtaining the fraction of empty bit map entries V_n) and plugging the result into the following equation (we derive Eq. (2) in Section 3.2):

$$\hat{n} = -m \ln V_n. \quad \text{(The symbol } \hat{} \text{ denotes an estimator.)} \tag{2}$$

The algorithm is made precise in Figure 1.

Figure 2 illustrates this process for one column of a relation having twelve tuples. In Figure 2, if there were no collisions, the count of "1" bits would be eleven, which is the desired column cardinality. Due to collisions, however, the measured count is six. Since the bit map has eight entries, the fraction of empty bit map entries is 2/8. Plugging this fraction into Eq. (2) gives an estimated column cardinality of 11.1.

3. ANALYSIS

In this section we derive theoretical values for the bias and standard error of the ratio $\hat{n}/n$. To make the analysis easier to follow, we first recast the problem of estimating the number of unique values in a column as the urn problem [13, 19]. Suppose that there are q balls and that each ball has one of n distinct colors. Balls are randomly assigned to a set of m urns in such a way that balls with the same color are assigned to the same urn. Then, linear counting can be transformed to the urn problem by associating tuples with balls, unique values with distinct colors, and bit map entries with urns. The terminology associated with this transformation, as well as additional notation, is summarized in Table I. Incidentally, this problem can also be cast as the one of counting the set cardinality (number of unique elements) from a multiset (with duplicates).

Algorithm Basic Linear Counting:

Let key_i = the key for the ith tuple in the relation.
Initialize the bit map to "0"s.
for $i = 1$ to q do
  hash_value = hash(key_i)
  bit map(hash_value) = "1"
end for
U_n = number of "0"s in the bit map
$V_n = U_n/m$
$\hat{n} = -m \ln V_n$

Fig. 1. Linear counting: The basic algorithm.

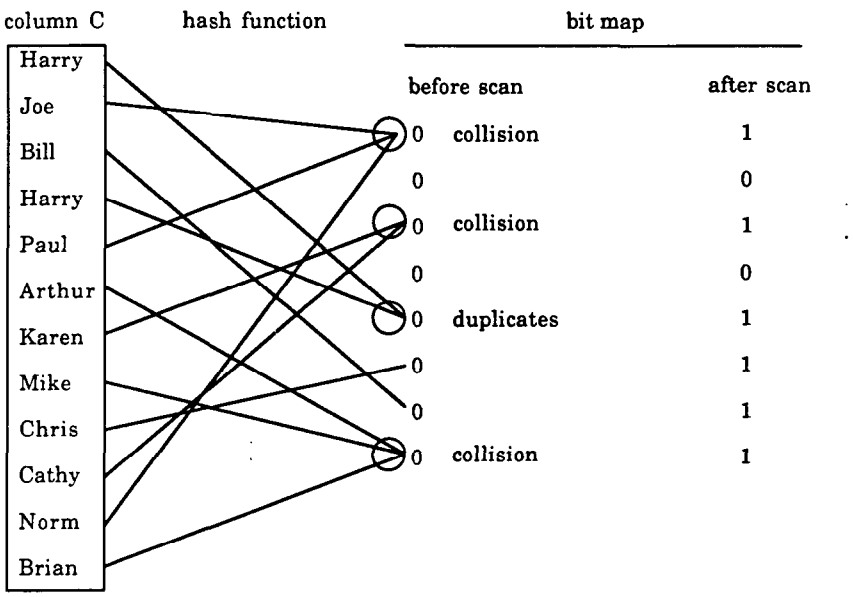

Fig. 2. Mapping a column of a relation into a bit map.

3.1 Properties of the Random Variables U_n and V_n

We summarize the formulas for the expectation and variance of U_n. Detailed derivation is presented in Appendix A:

$$E(U_n) = me^{-n/m} = me^{-t}, \quad \text{as} \quad m, n \to \infty, \tag{3}$$

$$\text{Var}(U_n) = me^{-t}(1 - (1 + t)e^{-t}), \quad \text{as} \quad m, n \to \infty. \tag{4}$$

Since $V_n = U_n/m$,

$$E(V_n) = e^{-t}, \quad \text{as} \quad m, n \to \infty, \tag{5}$$

$$\text{Var}(V_n) = \frac{1}{m} e^{-t}(1 - (1 + t)e^{-t}), \quad \text{as} \quad m, n \to \infty. \tag{6}$$

Table I. Terminology and Notation

Urn problem	Linear counting
n: Number of distinct colors	True count of unique values in the column (column cardinality)
q: Number of balls	Total number of tuples in the relation or total number of values in the column including duplicates (relation cardinality)
m: Number of urns	Number of bits in the bit map
t: n/m	n/m = load factor
U_n: Random variable denoting the number of empty urns	Random variable denoting the number of "0" bits after the test
V_n: U_n/m-random variable denoting the fraction of empty urns	Random variable denoting the fraction of "0" bits after the test
p: mean of V_n	
$\hat{n} = -m \ln V_n$: maximum likelihood estimator of n (see Appendix B)	

3.2 Derivation of the Estimator for the Number of Distinct Colors n

From Eq. (5), $E(V_n) = e^{-n/m}$, as $m, n \rightarrow \infty$. Replacing $E(V_n)$ and n by their representations in terms of observed variables, V_n and $\hat{n}$, we obtain

$$\hat{n} = -m \ln V_n, \tag{7}$$

where $\hat{n}$ is our estimator for n. Theorem A4 in Appendix B shows that $\hat{n}$ is the maximum likelihood estimator for n.

3.3 Derivation of the Bias

We derive the bias for the ratio $\hat{n}/n$ as follows. First, we expand the right hand side of Eq. (7) by its Taylor series about $p = E(V_n) = e^{-t}$, the mean of V_n. Denoting $-\ln(V_n)$ as $f(V_n)$,

$$\begin{aligned}\hat{n} &= m \times f(V_n) \\ &= m(f(p) + (V_n - p)f'(p) + \tfrac{1}{2}(V_n - p)^2 f''(p) \\ &\qquad + \tfrac{1}{6}(V_n - p)^3 f'''(p) + \tfrac{1}{24}(V_n - p)^4 f''''(p) \ldots) \qquad (8) \\ &= m\left(t - \frac{V_n - p}{p} + \frac{\frac{1}{2}(V_n - p)^2}{p^2} - \frac{\frac{1}{3}(V_n - p)^3}{p^3} + \frac{\frac{1}{4}(V_n - p)^4}{p^4} \ldots\right).\end{aligned}$$

We truncate Eq. (8) after the third term since the expected value of the second term is 0, and the third term is the first nonzero bias term. We shall discuss the error caused by this truncation in Appendix C. We then obtain the expected value of $\hat{n}$ as follows:

$$E(\hat{n}) = mt + \frac{m}{2p^2} E(V_n - p)^2.$$

Since, from Eq. (6), $E(V_n - p)^2 = \mathrm{Var}(V_n) = (1/m)e^{-t}(1 - (1 + t)e^{-t})$, we obtain

$$E(\hat{n}) = n + \frac{e^t - t - 1}{2}. \tag{9}$$

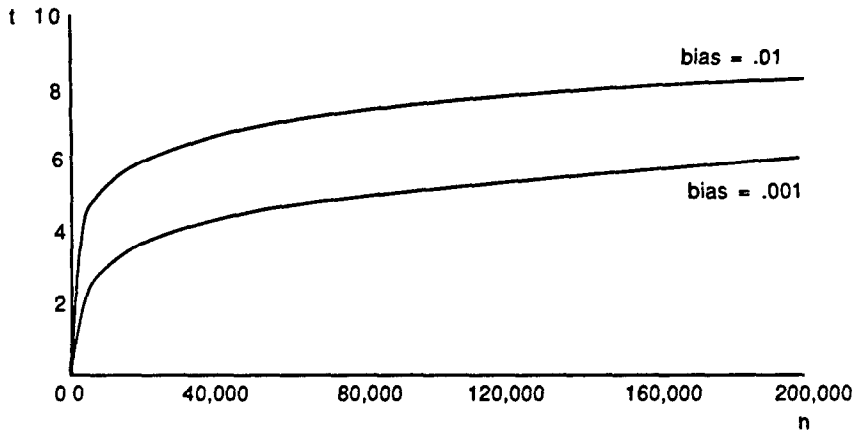

Fig. 3. Load factor that must be selected to achieve a specific bias.

Therefore,

$$\text{Bias}\left(\frac{\hat{n}}{n}\right) = E\left(\frac{\hat{n}}{n}\right) - 1 = \frac{e^t - t - 1}{2n}. \tag{10}$$

Notice that, given a constant load factor t, the estimator $\hat{n}/n$ is asymptotically unbiased. Figure 3 uses expression (10) to illustrate the relationship among the load factor t, the number of distinct colors n, and the bias. The graph shows the value of t that must be selected to achieve the desired bias for each value of n.

3.4 Derivation of the Standard Error

In this section we derive the standard error, that is, the standard deviation of the ratio $\hat{n}/n$. We proceed with the Taylor series expansion in the previous section. Truncating Eq. (8) after two terms, we obtain

$$\hat{n} = m\left(t - \frac{V_n - p}{p}\right). \tag{11}$$

The error analysis in Appendix C justifies this truncation. It follows that

$$\begin{aligned}
\text{Var}(\hat{n}/n) &= \frac{1}{n^2}\left(\frac{m^2}{p^2}\,\text{Var}(V_n - p)\right) \\
&= \frac{1}{n^2}\left(\frac{m^2}{p^2}\,\text{Var}(V_n)\right) \\
&= \frac{1}{n^2}\left(\frac{m^2}{p^2}\,\frac{1}{m}\right)p(1 - (1 + t)p) \qquad \text{from Eq. (6)} \\
&= \frac{m(e^t - t - 1)}{n^2}.
\end{aligned} \tag{12}$$

Hence, the standard error of the ratio $\hat{n}/n$ is as follows:

$$\text{StdError}\left(\frac{\hat{n}}{n}\right) = \frac{\sqrt{m}(e^t - t - 1)^{1/2}}{n}. \tag{13}$$

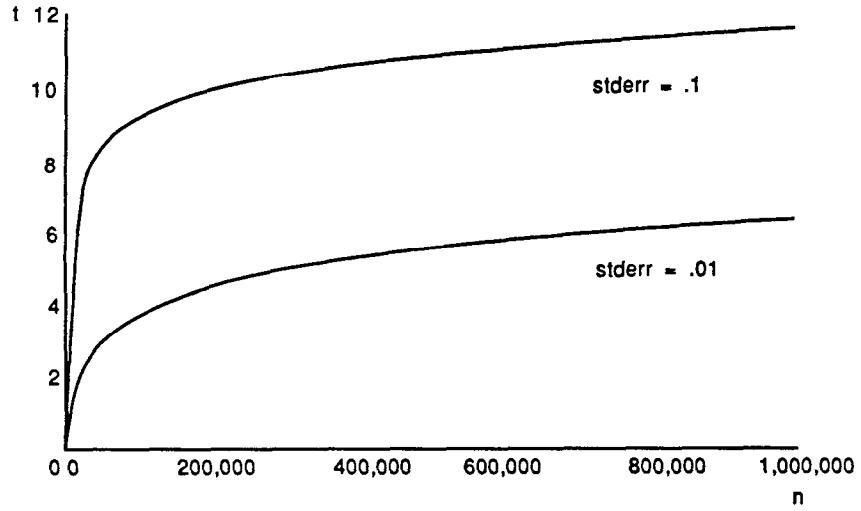

Fig. 4. Load factor that must be selected to achieve a specified standard error.

Table II. The Size of the Map Required to Assure that $m > \beta(e^t - t - 1)$ when $a = \sqrt{5}$

	Map size m epsilon			Map size m epsilon	
n	.01	.10	n	.01	.10
100	5034	80	80000	23029	10458
200	5067	106	90000	24897	11608
300	5100	129	100000	26729	12744
400	5133	151	200000	43710	23633
500	5166	172	300000	59264	33992
600	5199	192	400000	73999	44032
700	5231	212	500000	88175	53848
800	5264	231	600000	101932	63492
900	5296	249	700000	115359	72997
1000	5329	268	800000	128514	82387
2000	5647	441	900000	141441	91677
3000	5957	618	1000000	154171	100880
4000	6260	786	2000000	274328	189682
5000	6556	948	3000000	386798	274857
6000	6847	1106	4000000	494794	357829
7000	7132	1261	5000000	599692	439233
8000	7412	1412	6000000	702246	519429
9000	7688	1562	7000000	802931	598645
10000	7960	1709	8000000	902069	677040
20000	10506	3105	9000000	999894	754732
30000	12839	4417	10000000	1096582	831809
40000	15036	5680	50000000	4584297	3699768
50000	17134	6909	100000000	8571013	7061760
60000	19156	8112	120000000	10112529	8373376
70000	21117	9294			

Figure 4 uses Eq. (13) to show the relationship between the load factor t, the number of distinct colors n, and the standard error of $\hat{n}/n$. For each value of n, it plots the corresponding value of t that must be selected to achieve the desired standard error. Notice that as the number of distinct colors (column cardinality) increases, the load factor can be increased without increasing the standard error. The reason for this behavior is that if t is held constant as n approaches infinity, the standard error approaches zero. Thus, a constant standard error can be maintained by slowly increasing the load factor as n increases. This finding suggests that load factors much larger than one can achieve good accuracy. We explore this finding more rigorously in Section 4 (see Table II).

4. CONSTRAINTS RELATING THE MAP SIZE, LOAD FACTOR, AND STANDARD ERROR

Since linear counting is a probabilistic algorithm, there are a number of parameters that the user can manipulate to influence its performance. The first of these parameters is the standard error which, as noted in the introduction, is a measure of variability of the estimate provided by linear counting. Decreasing the standard error results in more precise estimates, but increases the required map size. Similarly, increasing the standard error results in less precise estimates, but decreases the required map size. The second parameter is the probability that the map becomes full. When the map is full, the experiment is fatally distorted because our estimate $\hat{n} = -m \ln V_n$ blows up. Thus, it is desirable to select a map size so that the probability that the map becomes full is negligible.

4.1 Two Constraints

Suppose the user wants to limit the standard error to ϵ. Slightly reformulating Eq. (13), this constraint can be written as

$$\frac{((e^t - t - 1)/m)^{1/2}}{t} < \epsilon,$$

or equivalently as

$$m > \frac{e^t - t - 1}{(\epsilon t)^2}. \tag{14}$$

Next, the map should be made sufficiently large to make the probability of its becoming full negligible. Let us note that, unless $m > n$ (i.e., $t < 1$), there is a nonzero probability that the map will become full. We can control this probability by allocating a map based on the constraint that the mean number of empty urns must be a standard deviations from zero. This constraint can be written as

$$E(U_n) - a \times \text{StdDev}(U_n) > 0. \tag{15}$$

Since $E(U_n) = me^{-t}$ and $\text{StdDev}(U_n) = (me^{-t}(1 - (1 + t)e^{-t}))^{1/2}$ (from Eq. (3) and Eq. (4)), Eq. (15) can be written as

$$me^{-t} > a(me^{-t}(1 - (1 + t)e^{-t}))^{1/2} \quad \text{or} \quad m > a^2(e^t - t - 1). \tag{16}$$

As we discuss in Section 4.2, $a = \sqrt{5}$ limits the probability of the map becoming full to 0.7 percent. From Eq. (14) and Eq. (16), we obtain

$$m > \beta(e^t - t - 1), \tag{17}$$

where $\beta = \max(a^2, 1/(\epsilon t)^2)$.

4.2 Probability that the Map is Filled Up

In this section, we discuss the probability of the map becoming full (*fill-up probability*) and the effect of β (more specifically, a) on this probability. In particular, we show that if $a > \sqrt{5}$ (meaning that $E(U_n)$ must be at least $\sqrt{5}$ standard deviations away from 0), then the fill-up probability is less than 0.7 percent.

We first introduce a lemma showing that the distribution of U_n converges to a Poisson distribution. (In Appendix B, we show that the distribution also converges to a normal distribution. However, in this section, we would like to use a discrete distribution that assigns a nonzero probability to the event that the map is filled up. Since the normal distribution is a continuous distribution, it assigns only an infinitesimal probability to this event. In principle, we can use a cumulative distribution for the interval [−0.5, 0.5)).

We then show that if $E(U_n) > \sqrt{5} \times \text{StdDev}(U_n)$, then $\text{Prob}(U_n = 0) < 0.007$.

LEMMA 1 [5]. *The limiting distribution of U_n, the number of empty urns, is Poisson with the expected value λ if*

$$me^{-n/m} \to \lambda \quad \text{as} \quad n, m \to \infty.$$

Thus,

$$\lim_{n,m\to\infty} Pr(U_n = k) = (\lambda^k/k!)e^{-\lambda}. \tag{18}$$

(Note that $E(U_n) = me^{-n/m} = \lambda$). The fill-up probability is then obtained as

$$Pr(U_n = 0) = e^{-\lambda}.$$

If $a > \sqrt{5}$, that is, $E(U_n) > \sqrt{5}$ StdDev(U_n), then $\lambda > \sqrt{5\lambda}$, or equivalently, $\lambda > 5$ since $E(U_n) = \lambda$ and StdDev$(U_n) = \sqrt{\lambda}$. Let us note that, in a Poisson distribution, mean = variance. Since $\lambda > 5$,

$$Pr(U_n = 0) < e^{-5} = .007(0.7\%).$$

This analysis shows an interesting phenomenon. Since $\lambda = E(U_n)$, the mean number of empty urns, the condition $\lambda > 5$ indicates that the (absolute) number of empty urns must be more than 5 to guarantee that the map is not filled more than 0.7 percent of the cases. Interestingly, this number is independent of n or m.

4.3 A Table To Assist Users in Choosing the Map Size

In Section 4.1 we obtained the constraint

$$m > \beta(e^t - t - 1) = \max(a^2, 1/(\epsilon t)^2) \times (e^t - t - 1). \tag{19}$$

Given n, Eq. (19) cannot be solved for m analytically since $t = n/m$ and m appears as both a linear and an exponential term. In this section we provide a

table of precomputed solutions of Eq. (19) to assist users in choosing the map size for the standard errors of .01 (1%) and .1 (10%) when $a = \sqrt{5}$ (i.e., the fill-up probability is 0.7%). We use the method of bisection [3]; the results are displayed in Table II. To use the table, locate the interval that brackets n and use interpolation to find the appropriate value of m. In Table II we note that the term $1/(\epsilon t)^2$ determines β for $\epsilon = .01$ while the term a^2 dominates when $\epsilon = .1$ and $n > 1000$. The term $1/(\epsilon t)^2$ dominates for $\epsilon = .1$ and $n \leq 1000$.

5. EXPERIMENTAL RESULTS

To verify the correctness of the theoretical derivations and analysis, we conducted a series of experiments, in which the estimators for the bias and standard error were compared with empirically derived figures. The map size for the experiments was varied between 100 and 100,000 and the loads were varied between .25 and 10 in the following manner:

map size: 100, 1000, 10000, 1000000
load factor: .25, .50, .75, 1, 2, . . . , 10

Each combination of these parameters was tested one hundred times. The experiments were performed by generating random numbers using a random number generator to simulate the set of unique values. Specifically, the experiments were organized as follows:

(1) Pick a map size m and a load factor t. Generate $n = m \times t$ random numbers between 0 and 1. n is the column cardinality we try to measure.
(2) Hash these numbers into the map by multiplying each random number by the map size.
(3) Count the number of "0" bits (U_n) in the map.
(4) Transform the measured value according to the maximum likelihood estimator $\hat{n} = -m \ln U_n/m$.
(5) Perform 100 such experiments for each data point and calculate the mean and the standard deviation of the estimator.
(6) Compare these results with the theoretical bias and standard error.

The results of the experiments are shown in Tables III–VI. Given a map size, we increased the load factor only up to the point where the map becomes full at least once during the experiments. As we predicted, these experimental points approximately coincided with the requirement $\beta \geq 5$ to keep the fill-up probability below 1 percent. (This can be checked by calculating $m/(e^t - t - 1)$ (see Eq. (19)) for the maximum load factor t that did not cause the map to be filled up.)

The experimental standard error and the theoretically predicted standard error were extremely close in almost every case. The experimental bias and theoretically predicted bias tended to diverge to some extent. However, the orders of magnitudes of these values stayed close to each other. We believe that this deviation is due to an insufficient number of random experiments. In fact, we observed that this deviation was reduced as we increased the number of random experiments. For example, when we increased the number of experiments to 1,000 (from 100), the experimental $\hat{n}/n$ for the case $m = 10{,}000$, $t = 4$, the

Table III. Map Size = 100

Load factor	Experimental $E[\hat{n}/n]$	Theoretical $E[\hat{n}/n]$	Experimental standard error	Theoretical standard error
0.25	0.995473	1.000681	0.069282	0.073784
0.50	0.999503	1.001487	0.074145	0.077129
0.75	1.006892	1.002447	0.076310	0.080774
1	1.012565	1.003591	0.092389	0.084752
2	1.027666	1.010973	0.117486	0.104750
3	1.016958	1.026809	0.138771	0.133689

Table IV. Map Size = 1000

Load factor	Experimental $E[\hat{n}/n]$	Theoretical $E[\hat{n}/n]$	Experimental standard error	Theoretical standard error
0.25	0.999197	1.000068	0.022948	0.023333
0.50	1.003729	1.000149	0.025523	0.024390
0.75	1.002573	1.000245	0.023620	0.025543
1	1.000577	1.000359	0.025823	0.026801
2	1.003372	1.001097	0.034144	0.033125
3	1.006535	1.002681	0.043413	0.042276
4	1.007168	1.006200	0.063848	0.055677
5	1.007115	1.014241	0.077536	0.075475

Table V. Map Size = 10000

Load factor	Experimental $E[\hat{n}/n]$	Theoretical $E[\hat{n}/n]$	Experimental standard error	Theoretical standard error
0.25	1.000175	1.000007	0.007149	0.007378
0.50	1.000392	1.000015	0.007504	0.007713
0.75	0.999416	1.000024	0.007576	0.008077
1	0.998791	1.000036	0.007927	0.008475
2	0.999860	1.000110	0.010844	0.010475
3	0.999488	1.000268	0.013565	0.013369
4	1.002690	1.000620	0.016362	0.017606
5	0.999464	1.001424	0.023171	0.023867
6	1.005000	1.003304	0.034150	0.033184
7	1.014530	1.007776	0.058061	0.047135

experimental value of $\hat{n}/n$ was 1.000755, which was very close to the theoretical prediction. Generally, since the bias is negligible, we can safely ignore this deviation. Taken as a whole, we observe an excellent match between the theoretical and experimental results.

6. LINEAR COUNTING: THE COMPLETE ALGORITHM

Figure 5 presents the complete linear counting algorithm.

Note that, in step 3 of the algorithm, we use the relation cardinality q instead of n since n is not known before the measurement is completed. Since $q \geq n$, generally, the algorithm will produce the result with a smaller standard error

Table VI. Map Size = 100000

Load factor	Experimental $E[\hat{n}/n]$	Theoretical $E[\hat{n}/n]$	Experimental standard error	Theoretical standard error
0.25	0.999977	1.000001	0.002096	0.002333
0.50	0.999874	1.000001	0.002256	0.002439
0.75	1.000235	1.000002	0.002894	0.002554
1	0.999992	1.000004	0.002726	0.002680
2	1.000241	1.000011	0.003658	0.003312
3	0.999878	1.000027	0.004317	0.004228
4	0.999918	1.000062	0.005681	0.005568
5	1.000011	1.000142	0.007748	0.007548
6	1.001603	1.000330	0.010973	0.010494
7	1.000730	1.000778	0.014368	0.014905
8	1.000408	1.001857	0.023627	0.021549
9	1.010030	1.004496	0.035842	0.031609
10	1.011620	1.011008	0.051350	0.046921

Algorithm Complete Linear Counting:

(1) Specify the desired accuracy (i.e., the standard error).

(2) Measure the relation cardinality (complexity = $O(q)$) if it is not already available.

(3) Read the map size from Table II using the relation cardinality q as n.

(4) Run the basic Linear Counting Algorithm.

(5) If the map fills up, rerun the basic Linear Counting Algorithm with a different hash function.

Fig. 5. Linear counting: The complete algorithm.

than specified. The relation cardinality is readily available in many systems because it is an important statistic that the system keeps track of. Otherwise, it can be measured by making only one pass over the relation. This process has $O(q)$ time complexity. In this case, the complete algorithm would require two passes over the relation. However, since more than one column can be processed simultaneously with this technique, the cost for each column can be kept lower than this. For example, for a relation having five columns, the average cost of obtaining the column cardinality of one column is that of making .4 pass over the relation. If the relation cardinality is already available, the cost would be only .2 pass over the relation.

When the load factor is greater than 1.0, there is a nonzero fill-up probability. However, this situation must occur very rarely since the fill-up probability is less than .7 percent. The probability that more than two iterations will be required is less than $.5 \times 10^{-4}$.

7. APPLICATIONS OF LINEAR COUNTING

As mentioned in Section 1, an important application of linear counting is the estimation of the column cardinality, which is one of the most important statistics that is used by many database systems. In addition to this application, linear counting has another interesting usage: namely, the estimation of the number of

distinct values in the join column resulting from an unconditional (i.e., without any selection predicates) join of two relations. This number (N_{UJ}) is an important parameter in estimating the size of a join result [4]. The role of this parameter is also demonstrated in [15], where the *join selectivity* $J(R, JP)$ of a relation R with respect to a *join path* JP is defined as N_{UJ}/N_{RJ}, where N_{RJ} is the number of distinct join column values of R. A *join path* is a set (R_1, $R_1.A$, R_2, $R_2.A$), where R_1 and R_2 are relations participating in the join and $R_1.A$ and $R_2.B$ are the join column of R_1 and R_2, respectively.

The number N_{UJ} is essentially the size of the set intersection between the join column of R_1 (JC_{R1}) and the join column of R_2 (JC_{R2}). To construct the set intersection of the two columns, either sorting or simple hashing can be used. However, they have the same performance problems as we mentioned in regard to obtaining the column cardinality. We present below an efficient algorithm to obtain N_{UJ} by slightly modifying linear counting. As in the case of the column cardinality, this algorithm requires only one or two scans over the relations involved depending on the availability of relation cardinalities. Thus, it will cause $c \times (q_1/b + q_2/b)$ disk accesses, where q_1 or q_2 is the relation cardinality of the relation R_1 or R_2 and c is either one or two.

The algorithm first obtains the set sizes of JC_{R1} and JC_{R2} using linear counting. It must use the bit maps of the same size for the two relations and must use the same hash function. Then, by applying logical OR to the two bit maps bit by bit, it constructs a third bit map reusing the storage of one of the original bit maps. From the third bit map, it calculates the set size of ($JC_{R1} \cup JC_{R2}$). Then, the set size of the intersection of the two columns is simply

$$\text{set\_size}(JC_{R1} \cap JC_{R2}) = \text{set\_size}(JC_{R1}) + \text{set\_size}(JC_{R2}) - \text{set\_size}(JC_{R1} \cup JC_{R2}).$$

Finally, the join selectivities of R_1 and R_2 with respect to the join path (JP) considered can be obtained as follows:

$$J(R_1, JP) = \text{set\_size}(JC_{R1} \cap JC_{R2})/\text{set\_size}(JC_{R1}),$$

$$J(R_2, JP) = \text{set\_size}(JC_{R1} \cap JC_{R2})/\text{set\_size}(JC_{R2}).$$

Figure 6 further illustrates the algorithm. In Figure 6, $\text{set\_size}(JC_{R1}) = -15 \times \ln 4/15 = 19.83$, $\text{set\_size}(JC_{R2}) = -15 \times \ln 6/15 = 13.74$, and $\text{set\_size}(JC_{R1} \cup JC_{R2}) = -15 \times \ln 3/15 = 24.14$. Hence, $\text{set\_size}(JC_{R1} \cup JC_{R2}) = 19.83 + 13.74 - 24.14 = 9.43$. The join selectivities are $9.43/19.83 = 0.48$ for R_1 and $9.43/13.74 = 0.69$ for R_2.

Let us note that logically ORing the two bit maps produces the third bit map as if it were constructed independently by hashing all the values in the columns JC_{R1} and JC_{R2}. Thus, using this map, we can measure the size of the union of the two columns.

Linear counting can be used to update the column cardinalities of the columns (especially those that do not have indexes) whenever all the statistics are updated. At the same time, the new algorithm can be used to update the join selectivities

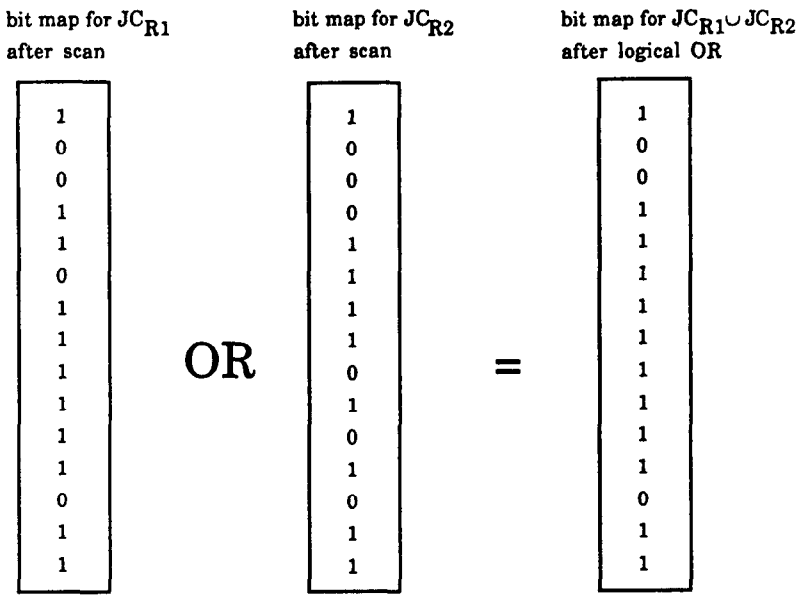

Fig. 6. Computing the join selectivity.

for the predetermined join paths. The set of join paths that are to be used frequently can be determined when the database is designed [16].

Finally, linear counting can also be used for measuring the cardinalities of composite columns (sets of multiple columns) by simply applying the technique to the composite key values from the component columns.

8. SUMMARY

We have presented a comprehensive theoretical and experimental analysis of linear counting—a probabilistic algorithm for counting the column cardinality—and proposed a complete algorithm that can be used in practical environments. The linear counting algorithm has $O(q)$ time complexity, where q is the number of tuples in the relation. The analysis has shown that we can achieve arbitrarily accurate estimation in linear time by prespecifying the standard error and using the appropriate map size.

We have analyzed two constraints relating the map size, the column cardinality, and the standard error. These constraints are due to (1) the standard error that the user prespecifies and (2) the probability that the map is filled up (fill-up probability). We have provided a table reflecting these constraints to assist users in choosing the proper map size.

A surprising result of the analysis is that a load factor much larger than 1.0 (e.g., 12) can be used for accurate estimation. With this property, when 10 million bits (1.25 Mbytes) of main memory is available, we find from Table II that linear counting can be profitably used with relations as large as 120 million tuples,

whereas it was indicated in [1] that linear counting can perform efficiently only up to 20 million tuples. For this number of tuples, we can achieve the standard error of 1 percent with the corresponding load factor of 12. In comparison, with the same amount of main memory available, simple hashing is efficient only up to 315 thousand tuples (Available Main Memory Size in bytes/Key Length in bytes) for integer keys and 62.5 thousand tuples for 20-character string keys. When compared with sorting, assuming $b = 1{,}000$, linear counting would require only 120,000 disk accesses, whereas a 10-way external sort merge would require 1.46 million disk accesses.

We have found an interesting phenomenon that the fill-up probability is determined by the absolute average number of "0" bits in the map regardless of the map size. In particular, an average of five or more "0" bits (from a 10-million-bit map, for example!) assures less than 0.7 percent of fill-up probability.

We have demonstrated two important applications of linear counting to database query optimization and physical database design. The first we have addressed is the estimation of column cardinality. The column cardinality is widely used to estimate the selectivities of predicates and the size of the answer sets of queries. The second is the estimation of the join selectivity. The join selectivity is a crucial parameter for estimating the size of a join result. We believe more applications can be found in the future as we gain experience with this technique. We believe our technique provides significant progress towards designing reliable database query optimizers and physical database design tools.

ACKNOWLEDGMENTS

We are indebted to Dina Bitton for bringing us together and for providing a forum in which the problems discussed in this paper were first broached. We wish to thank the referees for their very detailed comments that helped make the paper much more accurate and readable. Especially, we deeply appreciate a referee's suggestion to explore the application that we demonstrated in Section 7.

APPENDIX A: Properties of the Random Variable U_n

In this appendix we derive the mean and variance of the random variable U_n representing the number of empty urns.

Let A_j be the event that urn j is empty and let 1_{A_j} be the corresponding indicator random variable. Since the assignment of the balls is independent,

$$P(A_j) = \left(1 - \frac{1}{m}\right)^n$$

$$P(A_j \cap A_k) = \left(1 - \frac{2}{m}\right)^n, \qquad j \neq k.$$

Since U_n is the number of empty urns,

$$U_n = \sum_{j=1}^{m} 1_{A_j}.$$

Thus,

$$E(U_n) = \sum_{j=1}^{m} P(A_j) = m\left(1 - \frac{1}{m}\right)^n \cong me^{-n/m} = me^{-t}, \qquad \text{as} \quad n, m \to \infty$$

$$E(U_n^2) = E\left(\left(\sum_{j=1}^{m} 1_{A_j}\right)^2\right) = E\left(\sum_{j=1}^{m} 1_{A_j}^2\right) + 2E\left(\sum_{j=1}^{m}\sum_{i=1}^{j-1} 1_{A_i} 1_{A_j}\right)$$

$$= m\left(1 - \frac{1}{m}\right)^n + 2\left(\frac{m(m-1)}{2}\right)\left(1 - \frac{2}{m}\right)^n$$

since $(1_{A_j})^2 = 1_{A_j}$ and $1_{A_i} \times 1_{A_j} = 1_{A_i \cap A_j}$. Accordingly,

$$\begin{aligned}
\mathrm{Var}(U_n) &= E(U_n^2) - E(U_n)^2 \\
&= m\left(\left(1 - \frac{1}{m}\right)^n + (m-1)\left(1 - \frac{2}{m}\right)^n - m\left(1 - \frac{1}{m}\right)^{2n}\right) \\
&= m\left(\left(1 - \frac{1}{m}\right)^n - \left(1 - \frac{2}{m}\right)^n + m\left(\left(1 - \frac{2}{m}\right)^n - \left(1 - \frac{1}{m}\right)^{2n}\right)\right) \\
&\cong m(e^{-t} - e^{-2t} - te^{-2t}) \qquad \text{as} \quad m, n \to \infty \\
&= me^{-t}(1 - (1+t)e^{-t}).
\end{aligned}$$

In the derivation above, we used $m((1 - 2/m)^n - (1 - 1/m)^{2n}) \cong -te^{-2t}$ since

$$\begin{aligned}
\left(1 - \frac{2}{m}\right)^n &= \exp\left[n \ln\left(1 - \frac{2}{m}\right)\right] \\
&= \exp\left[n\left(-\frac{2}{m} - \frac{1}{2}\left(\frac{4}{m^2}\right) - \frac{1}{3}\left(\frac{8}{m^3}\right)\cdots\right)\right] \\
&= \exp\left[-\left(\frac{2n}{m}\right) \times \exp\left(-\frac{2n}{m^2}\right) \times \exp\left(-\frac{8}{3}\left(\frac{n}{m^3}\right)\right)\cdots\right]
\end{aligned}$$

$$\begin{aligned}
\left(1 - \frac{1}{m}\right)^{2n} &= \exp\left[2n \ln\left(1 - \frac{1}{m}\right)\right] \\
&= \exp\left[2n\left(-\frac{1}{m} - \frac{1}{2}\left(\frac{1}{m^2}\right) - \frac{1}{3}\left(\frac{1}{m^3}\right)\cdots\right)\right] \\
&= \exp\left(-\frac{2n}{m}\right) \times \exp\left(-\frac{n}{m^2}\right) \times \exp\left(-\frac{2}{3}\left(\frac{n}{m^3}\right)\right)\cdots
\end{aligned}$$

$$\begin{aligned}
m\left(\left(1 - \frac{2}{m}\right)^n - \left(1 - \frac{1}{m}\right)^{2n}\right) &\cong m \times \exp\left(-\frac{2n}{m}\right)\left(\exp\left(-\frac{2n}{m^2}\right) - \exp\left(-\frac{n}{m^2}\right)\right) \\
&\cong m \times \exp\left(-\frac{2n}{m}\right)\left(\left(1 - \frac{2n}{m^2}\right) - \left(1 - \frac{n}{m^2}\right)\right) \\
&\cong -te^{-2t} \qquad \text{as} \quad n, m \to \infty.
\end{aligned}$$

APPENDIX B: Approximating U_n as a Normal Variable

In this appendix we present a theorem showing that the distribution of U_n (and V_n) can be approximated as a normal distribution. We also show that $\hat{n}$ is the maximum likelihood estimator (MLE) of n.

THEOREM A1. *If $n, m \to \infty$ with $n/m \to t$ where t is a constant, then the limit distribution for the number of empty urns U_n is normal, that is, U_n is asymptotically normal.*

(Proof in [7, 12].)

COROLLARY A2. *The fraction of empty urns V_n is asymptotically normal.*

LEMMA A3. *Let $\hat{X}$ be the* MLE *of the variable X. If f is an invertible function* (i.e., *has a single-valued inverse*), *then $f(\hat{X})$ is an* MLE *of $f(X)$* ([8] Theorem 1, p. 284).

THEOREM A4. *$\hat{n}$ is the* MLE *of n.*

PROOF. Since V_n is a random sample taken from a normally distributed population, V_n is the MLE of the mean fraction of empty urns $E(V_n)$. Further, since $-\ln X$ is invertible, $\hat{n} = -m \ln V_n$ is the MLE for $-m \ln E(V_n) = n$. □

APPENDIX C: Error Bounds in Analysis

In Section 3 we derived the bias and standard error of the estimator $\hat{n}/n$ by truncating the Taylor series after the third and second terms respectively. The question naturally arises: How much error is introduced by this truncation? The answer is "not much." Specifically, for the bias, we show that the first nonzero remainder term truncated from the series is only $3/2\beta$ as large as the last term included in the series. For the variance (*standard error*^{2}), we show that the relative error caused by not including the third term is $1/2\beta$. Since $\beta \geq 5$ this would mean approximately 30 percent of error in the estimation of the bias and 5 percent of error in the estimation of the standard error (10% for the variance).

For the discussion of the error bounds, it is convenient to rewrite V_n in terms of the standard normal distribution (Appendix B shows that the distribution of V_n can be approximated as a normal distribution). Let ξ denote the standard normal ($N(0, 1)$) random variable and write

$$V_n = p + \sigma\xi, \tag{20}$$

where $p = E(V_n) = e^{-t}$ is the mean and $\sigma = ((1/m)e^{-t}(1 - (1 + t)e^{-t}))^{1/2}$ is the standard deviation of V_n. Substituting this value of V_n into Eq. (8), we obtain

$$\hat{n} = m\left(t - \frac{\sigma\xi}{p} + \frac{(\frac{1}{2})\sigma^2\xi^2}{p^2} - \frac{(\frac{1}{3})\sigma^3\xi^3}{p^3} + \frac{(\frac{1}{4})\sigma^4\xi^4}{p^4} \cdots\right). \tag{21}$$

C.1 Error Bound in Analysis of the Bias

We derive the error bound on the first nonzero remainder term in the Taylor series expansion for the bias $(E(\hat{n}/n) - 1)$. From Eq. (21)

$$E(\hat{n}) = mt + 0 + \left(\frac{m}{2}\right)\left(\frac{1}{m}\right)e^{-t}(1 - (1 + t)e^{-t})e^{2t} + 0$$
$$+ E(\xi^4)\left(\frac{m}{4}\right)\left(\frac{1}{m^2}\right)e^{-2t}(1 - (1 + t)e^{-t})^2e^{4t}$$
$$+ \text{higher-order terms.}$$

In Section 3 we truncated the series after the third term. Thus, the first nonzero remainder term for this approximation of $E(\hat{n}/n)$ is

$$\left(\frac{E(\xi^4)}{4m}\right)e^{-2t}(1 - (1 + t)e^{-t})^2e^{4t}.$$

The size of this term as a fraction of the last term used in the approximation is

$$\begin{aligned}
&(E(\xi^4)e^{-2t}/4m)(1 - (1 + t)e^{-t})^2e^{4t}/(e^{-t}/2)(1 - (1 + t)e^{-t})e^{2t} \\
&\quad = 3e^t(1 - (1 + t)e^{-t})/m \qquad (E(\xi^4) = 3 \text{ for a standard normal distribution}) \\
&\quad = 3(e^t - t - 1)/2m \\
&\quad < 3(e^t - t - 1)/2\beta(e^t - t - 1) \qquad (\text{since } m > \beta(e^t - t - 1)) \\
&\quad = 3/2\beta.
\end{aligned}$$

Since $\beta \geq 5$, the inclusion of additional terms in the Taylor series for $\hat{n}$ in the computation of $E(\hat{n})$ would increase the computed bias by approximately 30 percent. Given the generally negligible value of the bias (as we saw in Section 6), we decided that the inclusion of additional terms was not justified.

C.2 Error Bound in Analysis of the Standard Error

We derive the bound of the error caused by not including the third term of the Taylor series for $\hat{n}$ in computing the standard error for the estimator $\hat{n}/n$. In this derivation we first calculate $\Delta\ \mathrm{Var}(\hat{n}/n)$ when the third term of the series is added in the approximation, and then, take the ratio of the result over the variance that would be obtained with only two terms (i.e., Eq. (12)).

From Eq. (21), truncating after three terms, we obtain

$$\frac{\hat{n}}{n} = m\left(t - \frac{\sigma\xi}{p} + \frac{\frac{1}{2}\sigma^2\xi^2}{p^2}\right).$$

The resulting expression for $\mathrm{Var}(\hat{n}/n)$ is

$$\begin{aligned}
\mathrm{Var}\left(\frac{\hat{n}}{n}\right) &= \left(\frac{1}{n^2}\right)\mathrm{Var}\left(m\left(t - \frac{\sigma\xi}{p} + \frac{\frac{1}{2}\sigma^2\xi^2}{p^2}\right)\right) \\
&= \left(\frac{m}{n}\right)^2\left(\mathrm{Var}(t) + \left(\frac{\sigma}{p}\right)^2\mathrm{Var}(\xi) + \frac{1}{4}\left(\frac{\sigma}{p}\right)^4\mathrm{Var}(\xi^2)\right) \qquad (22) \\
&\quad - 2\left(\frac{m}{n}\right)^2\frac{1}{2}\left(\frac{\sigma}{p}\right)^3\mathrm{Cov}(\xi, \xi^2),
\end{aligned}$$

where Cov is the covariance of the two random variables. To avoid clutter, two covariance expressions that are zero, $\mathrm{Cov}(t, \xi)$ and $\mathrm{Cov}(t, \xi^2)$ are omitted. $\mathrm{Cov}(\xi, \xi^2)$ is also zero because

$$\begin{aligned}\mathrm{Cov}(\xi, \xi^2) &= E(\xi \times \xi^2) - E(\xi) \times E(\xi^2) \qquad [8]\\ &= E(\xi^3) - 0 \qquad (\text{since } E(\xi) = 0)\\ &= 0 \qquad (\text{since } E(\xi^3) = 0 \text{ if } \xi \text{ is a standard normal variable}).\end{aligned}$$

Further, $\mathrm{Var}(t) = 0$ since t is a constant. Using $\mathrm{Var}(\xi) = 1$, the variance of $\hat{n}/n$ can be simply written as

$$\mathrm{Var}\left(\frac{\hat{n}}{n}\right) = \left(\frac{m}{n}\right)^2\left(\frac{\sigma}{p}\right)^2 + \frac{1}{4}\left(\frac{m}{n}\right)^2\left(\frac{\sigma}{p}\right)^4 \mathrm{Var}(\xi^2). \tag{23}$$

$\mathrm{Var}(\xi^2) = 2$ since ξ^2 has a Chi-square distribution with one degree of freedom. (If χ has $N(0, 1)$, then χ^2 has $\chi^2(1)$. Since the variance of a Chi-square distribution is two times its degree of freedom, $\mathrm{Var}(\xi^2) = 2$.)

Substituting σ and p with its values, we note that the first term of Eq. (23) is identical to Eq. (12), which would be the computed value of the variance if only two terms of the series were considered. Thus, the bound of the error caused by excluding the third term is

$$\frac{1}{4}\left(\frac{m}{n}\right)^2\left(\frac{\sigma}{p}\right)^4 \mathrm{Var}(\xi^2) \Big/ \left(\frac{m}{n}\right)^2\left(\frac{\sigma}{p}\right)^2 = \frac{1}{2}\left(\frac{\sigma}{p}\right)^2. \tag{24}$$

Substituting $\sigma = ((1/m)e^{-t}(1 - (1 + t)e^{-t}))^{1/2}$ and noting $m > \beta(e^t - t - 1)$, we transform Eq. (24) as

$$\begin{aligned}\frac{1}{2}\left(\frac{\sigma}{p}\right)^2 &= \left(\frac{1}{2m}\right)(e^t - t - 1)\\ &< \frac{e^t - t - 1}{2\beta(e^t - t - 1)}\\ &= \frac{1}{2\beta}.\end{aligned} \tag{25}$$

Since $\beta \geq 5$, the inclusion of the third term of the Taylor series for $\hat{n}$ in the computation of the variance adds at most 10 percent. Since we are actually interested in the standard error, we must take the square root of $(1 + 1/2\beta)$ to obtain the percentage difference between the two computations of the standard error. Since the square root of 1.1 is approximately 1.05, we would encounter an error of only 5 percent. Thus, the addition of the third term in the calculation of the standard error was not deemed worthwhile.

REFERENCES

1. ASTRAHAN, M., SCHKOLNICK, M., AND WHANG, K.-Y. Approximating the number of unique values of an attribute without sorting. *Inf. Syst. 12*, 1 (1987), 11–15.
2. BITTON, D., AND DEWITT, D. J. Duplicate record elimination in large data files. *ACM Trans. Database Syst. 8*, 2 (June 1983), 255–265.
3. CONTE, S. D., AND DE BOOR, C. *Elementary Numerical Analysis: An Algorithmic Approach.* 3rd ed., McGraw-Hill, New York, 1980.

4. DEMOLOMBE, R. Estimation of the number of tuples satisfying a query expressed in predicate calculus language. In *Proceedings of the 6th International Conference on Very Large Data Bases*, 1980, pp. 55–63.
5. FELLER, W. *An Introduction to Probability Theory and Its Applications.* Vol. 1, 3rd ed., Wiley, New York, 1968.
6. GELENBE, R., AND GARDY, D. The size of projections of relations satisfying a functional dependency. In *Proceedings of the 8th International Conference on Very Large Data Bases* (Mexico City, 1982), pp. 325–333.
7. JOHNSON, N. L., AND KOTZ, S. *Urn Models and Their Application.* Wiley, New York, 1977.
8. MOOD, A. M., GRAYBILL, F. A., AND BOES, D. C. *Introduction to the Theory of Statistics.* 3rd ed., McGraw-Hill, New York, 1974.
9. OLKEN, F., AND ROTEM, D. Simple random sampling from relational databases. In *Proceedings of the 12th International Conference on Very Large Data Bases* (Kyoto, Aug. 1986), pp. 160–169.
10. ROWE, N. C. Antisampling for estimation: An overview. *IEEE Trans. Softw. Eng. SE-11*, 10 (Oct. 1985), 1081–1091.
11. SELINGER, P., ET AL. Access path selection in a relational database management system. In *Proceedings of the International Conference on Management of Data, ACM SIGMOD*, 1979, pp. 23–34.
12. SHERMAN, B. A random variable related to the spacing of sample values. *Ann. Math. Statistics 21* (1950), 339–361.
13. VANDER-ZANDEN, B. T., TAYLOR, H. M., AND BITTON, D. Estimating block accesses when attributes are correlated. In *Proceedings of the 12th International Conference on Very Large Data Bases* (Kyoto, Aug. 1986), pp. 119–127.
14. WHANG, K., WIEDERHOLD, G., AND SAGALOWICZ, D. Estimating block accesses in database organizations: A closed noniterative formula. *Commum. ACM, 26*, 11 (Nov. 1983), 940–944.
15. WHANG, K., WIEDERHOLD, G., AND SAGALOWICZ, D. Separability—An approach to physical database design. *IEEE Trans. Comput. C-33*, 3 (Mar. 1984), 209–222.
16. WIEDERHOLD, G., AND EL-MARSI, R. The structural model for database design. In *Proceedings of the International Conference on Entity Relationship Approach* (Los Angeles, Dec. 1979), pp. 247–267.
17. WIEDERHOLD, G. *Database Design.* 2nd ed., McGraw-Hill, New York, 1983.
18. YOUSSEFI, K., AND WONG, E. Query processing in a relational database management system. In *Proceedings of the 5th International Conference on Very Large Data Bases*, 1979, pp. 409–417.
19. YU, C. T., LAM, K., SIU, M. K., AND OZSOYOGLU, M. Performance analysis of three related assignment problems. In *Proceedings of the International Conference on Management of Data, ACM SIGMOD*, 1979, pp. 82–92.

Received August 1987; revised July 1988 and February 1989; accepted April 1989