Selectivity Estimation Using Moments and Density Functions

Goetz Graefe

Oregon Graduate Center
Department of Computer Science
and Engineering
19600 N.W. von Neumann Drive
Beaverton, OR 97006-1999

Technical Report No. CS/E 87-012
November 1987

Selectivity Estimation using Moments and Density Functions

Goetz Graefe
Oregon Graduate Center

Abstract

A concise description of the distribution of attribute values is essential in database query optimization to estimate the selectivity of database operations and the sizes of intermediate results of a query. Most current methods used to estimate result sizes depend on assumptions that are rarely justified in real databases, namely the assumptions of uniform distribution of each attribute and statistical independence between attributes. Moments and density functions are used in statistics to describe the distribution of a population. Compared to histograms which are used in some database systems to describe value distributions, moments and density functions offer the advantages that they require less storage space and that they can be updated much more efficiently. Statistical dependencies between attributes can be described using co-moments and multi-dimensional density functions, allowing for accurate estimation of complex predicates and combinations of selection and join predicates.

1. Introduction

Database query optimization is the task of finding the optimal query execution plan for a given query. A large number of strategies to be used in query optimization have been reported in the database literature. For a recent survey, see [Jarke1984a]. Most of these proposals and implementations, however, do not adequately address the problem of estimating the sizes of intermediate results. To understand the significance of the problem, consider the following relational query.

> Find all employees with a salary between $20,000 and $30,000 and the name of their departments.

To evaluate this query, the database system must perform a selection on the employee relation, and join the result with the department relation. Assume that each employee record includes

the id of the employee's department, and that the department file is indexed on id's[1]. This index can be used very effectively to perform the join operation [Blasgen1977a]. Using the index, only those records and pages from the department file which contain relevant department records must be read from disk. However, if many departments have employees in this salary range, probably all pages of the department file must be read. Since the department records are requested in random order (actually in the order in which the employee file is scanned) each page may be needed several times. Depending on the buffer size and replacement strategy, each page may be read from disk several times, clearly a very undesirable situation.

In order to decide on the optimal processing strategy in a situation like the one described in this example, the database system needs to anticipate as correctly as possible how many records from each file will actually be needed from each file to evaluate a query. This frequently requires estimates of intermediate result sizes, in the example the result of the selection. We propose to investigate the use of statistical moments and density functions as a basis for more reliable and more accurate estimations.

The problem of estimating intermediate result sizes is not restricted to relational databases. In fact, the problem arises in all database systems that support complex queries on sets of objects. In more intelligent database systems, queries will require more operations to evaluate, and optimization need and opportunity will be greater. Since the reliability of estimates decreases with the number of processing steps, improved estimation procedures are very important.

In the next section, estimation methods used or proposed previously are described. In Section 3, we introduce moments and density functions as they are used in statistics to describe data distributions. Section 4 outlines how density functions can be used to estimate the number

---

[1] We assume in this example that each relation is stored in its own file, and that a disk page belongs to one file only. We use the words *relation* and *tuple* when we refer to the conceptual level, and the words *file* and *record* for the physical level.

of records from one file satisfying a complex condition. In Section 5, estimation procedures for joins are presented. Section 6 describes the use of density functions to estimate the result size of projections and aggregate functions. Section 7 shows some preliminary results using graphs to compare a density function with an approximation density function calculated using random samples and moments. In Section 8, we show how moments can be collected and maintained very efficiently in database environments. Section 9 contains a summary and our conclusions.

2. Previous Work

When considering the large amount of research that has been done on database query optimization, it is surprising how relatively few research reports have dealt with estimating the size of intermediate results.

In the original INGRES effort [Stonebraker1976a], the difficulty to anticipate results sizes led to the development of the query processing algorithm that interleaves query optimization and execution [Wong1976a]. Each processing step produced a temporary relation, the size of which was exactly known in the next optimization step. Besides the fact that this approach has clear disadvantages when a query runs many times in virtually the same environment (e.g. a banking teller transaction), this algorithm can miss the optimal query execution strategy if the result of a processing step is significantly larger than expected.

In System R, information from existing indices was used as far as possible to estimate the result size of a single relation query, namely cardinality, key cardinality (number of distinct values), minimum attribute value, and maximum attribute value [Selinger1979a]. If no suitable index existed, a set of "magic" constants was used to estimate the selectivity of a predicate, i.e. the fraction of qualifying tuples. For predicates of the form $attribute = constant$, the selectivity was set to $\frac{1}{key\ cardinality}$; if the key cardinality was unknown, 10%. For a predicate of the form $attribute < constant$, the selectivity was set to $\frac{constant - minimum}{maximum - minimum}$; if minimum and maximum attribute value are unknown, 33%. This formula is based on the assumption that the

attribute values are uniformly distributed. For predicate involving the Boolean operators *AND* and *OR*, it was assumed that attributes are independently distributed, e.g. the selectivity of the conjunction of two predicates is set to be the product of the individual selectivities.

These two assumptions, uniform distribution for each attribute and independent distribution of each pair of attributes, are frequently not met in real databases. Consider, for example, a relation of employees which includes attributes for salaries and tax withholding. The salaries are probably not uniformly distributed from \$0 (for a volunteer) to \$100,000 (for the CEO), and the salary and the tax withholding are certainly not independent. If a query predicate includes restrictions on these two attributes, i.e. salary $>$ \$50,000 and tax $<$ \$1,000, the formulas used in System R are bound to give incorrect estimates[2].

Uniqueness of keys can be used in determining the worst case (largest) selectivity of select, project, and join operators. This technique has been used both in System R [Selinger1979a] and in INGRES [Epstein1979a]. Furthermore, functional dependencies can also be incorporate in the estimation procedure.

To improve the accuracy of estimates for selections, histograms were implemented in the commercial version of INGRES [Kooi1982a]. Within each interval of the histogram, a uniform distribution of values is assumed. If a histogram is sufficiently detailed, this assumption does not have a significant impact. There are two problems with histograms. First, they do not work well if the distribution is very uneven. Consider the distribution of salaries and estimations using a histogram with 10 intervals. There are probably many more employees with salaries in the range of \$25,000 to \$30,000 than in the range \$95,000 to \$100,000. The estimated selectivity for the predicate *salary* $>$ *\$97,500* is very accurate, but the estimate for *salary* $>$ *\$27,500 and salary* $<$ *32,500* is subject to significant error. Inverted histograms have been

[2] We would like to investigate the influence of incorrect estimates on the optimality of query execution plans. However, we view the "stability" of access plans as a different research topic [Graefe1987a].

suggested to deal with this difficulty [Piatetsky-Shapiro1984a]. Instead of using counts for intervals of equal width, the limits of intervals with equal counts are used[3]. The major problem with inverted histograms is how to find and to update them efficiently because this requires sorting the data values. The second problem with histograms is that they do not address the independence assumption. To our knowledge, work in progress by Muralikrishna at the University of Wisconsin - Madison is the first attempt to use multi-dimensional histograms.

Other research efforts were directed to estimating the number of disk blocks that must be accessed to retrieve all relevant records for a query. If the distribution of records over disk blocks is unrelated to attributes in the query predicate, blocks are accessed virtually at random, and Yao's formula [Yao1979a] is an appropriate way to estimate block accesses. If the attributes in the query predicate are correlated with the clustering attribute (i.e. records are assigned to disk blocks according to some attribute value), the estimation becomes fairly complex [Zanden1986a].

Statistical concepts were used by Christodoulakis [Christodoulakis1983a] to estimate the number of records satisfying a condition. He assumed that the data values in each attribute followed one of three parameterized uni-modal distributions, and maintained a covariance matrix for each relation, used to determine the correlation between pairs of attributes. Our work differs from his in several respects. First, we do not assume that the data follow a predefined distribution. Our approach allows us to approximate any data distribution. Second, the accuracy of estimations in our approach depends on the effort spent: The more moments are gathered, the more accurate will the estimation be. Third, we will use our techniques to estimate the result sizes of join, project, and aggregate functions, too. Fourth, we will be able to estimate the attribute distributions in result relations, which is particularly useful for intermediate results.

---

[3] In statistics, these limits are called *quantiles*. The best known special case of quantiles is the median, which is the 50% quantile.

Yang [Yang1985a] derived reliable formulas to estimate the cardinality of a join result. The expected size is the product of the input cardinalities divided by the cardinality of the join domain; to calculate the exact result cardinality, a correction term must be added. The correction term is calculated from the number of occurrences (frequency counts) of each value in each of the join columns. It is the product of the two standard deviations of the frequency counts and the correlation coefficient between them. The correction term can easily be much larger than the term for the expected size, hence it is important to calculate or estimate it accurately. Unfortunately, maintaining the frequency counts is very expensive for a large database. Yang suggests to combine partitioning, approximation, and sampling methods for periodic updates of the statistical information, but gives no report on practical experiences with these methods. A possibly more important drawback of the estimation procedures proposed by Yang is their inability to deal with databases and queries involving multiple operators and correlated attributes. For example, if a relation contains the (heavily correlated) attributes *salary* and *tax withholding*, and a query requires a selection according to *salaries* and a join according to *tax withholding*, Yang's methods do not help because it is not possible to capture this correlation in the statistics used for estimating the join size. Nevertheless, we intend to use as much of Yang's work as possible by appropriately adapting and extending the formulas provided.

3. Statistical Moments

A moment is a sum of the data values in a distribution raised to a certain power. For example, for the sequence $x_1, x_2, ..., x_N$, the k^{th} moment is

$$m_k = \sum_{i=1}^{N} x_i^k .$$

The first moment is simply the sum of the values. The second moment is the sum of the squares. The zero-th moment can be defined as the number of values in the sequence. The first and second moment can be used to calculate the variance of a population, because

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} = \frac{m_1}{m_0}$$

$$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N} = \frac{\sum_{i=1}^{N} x_i^2 - 2\sum_{i=1}^{N} x_i \bar{x} + N\bar{x}^2}{N} = \frac{m_2 - \frac{m_1^2}{m_0}}{m_0} = \frac{m_2}{m_0} - (\frac{m_1}{m_0})^2$$

In fact, if one were to write a program to find the mean and the variance of a long vector, one would intuitively choose to use the first and second moments.

Co-moments describe the distribution of more than one variable. The k,l^{th} moment of the sequences x_i and y_i is defined as

$$m_{k,l} = \sum_{i=1}^{N} x_i^k y_i^l$$

Co-moments can be used to calculate the covariance, correlation, and the regression constants. Co-moments of more than two variables are defined analogously.

Another method used in statistics to describe distributions are distribution and density functions. For each value in the data domain (e.g. *salaries*) the distribution function expresses what portion of the data are equal to or less than the given value. The value of the distribution function of arguments less than the minimum is 0, of arguments equal to or greater than the maximum, it is 1. The density function is the derivative of the distribution function, thus expressing how likely a certain value is to occur. Probably the best known density function is the bell-shaped curve of the normal distribution.

For multi-dimensional distributions, distribution functions and density functions can be defined to map a pair of values (triple, quadruple, etc., depending on the number of dimensions) to the fraction of data pairs (triples, etc.) with all values less than or equal than the given ones.

If the form of the distribution is known, e.g. uniform, normal, exponential, etc., the exact distribution can easily be determined from suitable number of moments (typically two). If the form of the distribution is not known, the distribution can nevertheless be approximated by

assuming that the density function is of a particular form. It is important that this assumption is not restrictive, i.e. that the form assumed is able to model any density function. We suggest using polynomials which allow to model density functions with the desired accuracy by choosing the degrees of the polynomials sufficiently high. Furthermore, polynomials offer the advantage that it is very easy to find the distribution function from the density function, and vice versa.

The coefficients of the density function can be calculated from the moments. Incidentally, the number of coefficients that can be calculated from a set of moments is exactly the number of moments available. For example, if we intend to describe the density function with a polynomial of degree 3, we need to know the moments m_1, m_2, m_3, and the cardinality (m_0).

Polynomials of degree 3 or 5 will probably be sufficient to model many density functions in real databases with appropriate accuracy. In the section on implementation plans, we argue that collecting and maintaining moments can be done very efficiently, such that 3 or 5 moments for each attribute are not at all unrealistic.

Calculating polynomial coefficients from moments involves solving a (small) system of linear equations. Assume that we know the moments $m_1, m_2, \ldots, m_M$, the cardinality m_0, and the lower bound L and the upper bound U of a distribution. If we assume that the density function is of the form

$$f(x) = \sum_{j=0}^{M} a_j x^j$$

i.e. a polynomial of degree M, we can determine the coefficients a_j using a set of equations. For each moment m_i, $i=0,\ldots,M$, we know

$$m_i = \int_L^U x^i f(x)\,dx = \int_L^U x^i \sum_{j=0}^{M} a_j x^j\,dx = \sum_{j=0}^{M} a_j \int_L^U x^{i+j}\,dx = \sum_{j=0}^{M} a_j \frac{U^{i+j+1} - L^{i+j+1}}{i+j+1}$$

This is a system of linear $M+1$ equations with $M+1$ variables a_j, $j=0,\ldots,M$. The complexity of solving a system of linear equations is $O(N^3)$ for N equations; however, if the minimum L and maximum U do not change, the system can be stored in a triangularized form which allows to

determine the coefficients in $O(N^2)$ (see, for example [Cheney1980a]).

4. Estimating Selection Result Sizes

If the density function and all its parameters are known, it is easy to estimate the number of values in a given interval. In order to determine the number of tuples with an attribute between the lower limit L and the upper limit U, simply use the distribution function F or the integral of the density function f.

$$\int_L^U f(x)\,dx = F(U)-F(L)$$

The result of this formula and the following formulas must be scaled using the cardinality of the relation to find the result cardinality. In this paper, we omit this multiplication to keep the formulas clearer. If the distribution function and the density function are polynomials, the integral can be calculated particularly efficiently.

If the selection is an equality constraint on an non-unique attribute, the best guess under the uniformity assumption is that all values are distributed over all distinct keys, expressed in the formula

$$\frac{1}{key\ cardinality}$$

The number of distinct keys is usually determined and maintained within an index. If we can assume that the values in the domain are equally spaced, the density function contains more information about the distribution of values, thus we suggest modifying the last formula to

$$\frac{1}{|A|} * f(a_0) * (max-min)$$

$|A|$, the cardinality of the domain, is equal to the *key cardinality* used in existing query optimizers. a_0 is the constant from the query, thus $f(a_0)$ is the number of attribute values equal to a_0. max and min express the range of the data values, e.g. salaries in the range from \$0 to \$100,000. The correction factor $(max-min)$ is required because it is this range over which the integral of the density function is 1.

To estimate the result size of a selection with two attributes, e.g. *salary* > *$50,000 and tax* < *$1,000*, the two-dimensional density function must be calculated from the moments and co-moments (which can be done in the same manner as for single attribute distributions), and a double integral must be solved. For example, to find the number of tuples with $L_x \leq x < U_x$ and $L_y \leq y < U_y$, the value for

$$\int_{L_x}^{U_x} \int_{L_y}^{U_y} f(x,y)\, dy\, dx = F(U_x,U_y) - F(U_x,L_y) - F(L_x,U_y) + F(L_x,L_y)$$

must be found. As for one-attribute selections, this is particularly simple if the density function is a polynomial. Notice that these formulas can be expected to give very accurate estimates whether or not the attributes x and y are correlated.

If more than two attributes are involved in a selection predicate, the expressions become larger, but not more complex. Nevertheless, this procedure is more expensive to perform than procedures based on the assumption of statistical independence between attributes. Fortunately, the attributes and attribute combinations for which moments and density functions give significantly more accurate estimates can be determined automatically using the moments. As mentioned before, moments can be used to calculate the correlation between attributes. If the correlations between a set of attributes is known, it can be decided which attributes can be considered independent. It will be very interesting to see to which extent this can be done efficiently in a database system.

The density function can also be used to estimate the number of disk blocks that contain relevant tuples for a given query. Conceptually, we calculate the sum of the probabilities for the records in each block. We are confident that it will be relatively easy to design a closed form expression which eliminates the need for a explicit summation.

5. Estimating Join Result Sizes

Estimating the result size for join operations is considered substantially harder than for selection operators. The reason is that data about two relations are required. An exact

formula to estimate the result size was given by Yang [Yang1985a]. The dependency on frequency counts, however, makes this formula somewhat impractical, and no implementation experiences were reported. We believe that using moments, the formulas can be made to work for a dynamically changing, large database. Essentially, we plan to adapt the formulas to use continuous instead of discret distributions, as we did for selections.

The result size of an equi-join is the sum of the products of the frequency counts from the two relations for each value in the join domain. Let us assume the join domain is A with values a_k, k=1,2,...,$|A|$, and the frequency counts for these values are r_k and s_k when joining relation R and S. Notice that r_k and s_k are not the values of the join attributes in R and S; these values are represented by a's. Each value a_k in the join domain produces a number of tuples in the join result, namely the product of the two corresponding frequency counts r_k and s_k. The cardinality of the join result is the sum of these products, i.e.

$$\sum_{k=1}^{|A|} r_k s_k$$

Instead of using frequency counts, we suggest to use continuous density functions derived from the moments. For density functions f_R and f_S, the result of the join result is

$$\int_A f_R(a)\, f_S(a)\, da$$

If both of these density functions are polynomials, this integral can be solved very efficiently by combining the two polynomials of a into one.

Yang suggests applying the estimation procedure for the result size of a multi-attribute equi-join (i.e. a join with several equality clauses in the join predicate) by conceptualizing the join predicate as a single attribute join on the cross product of all join domains. Using co-moments and multi-dimensional density functions, this is very straightforward.

When one of the input relations of a join operator is the result of a select operator, estimating the result size is considered particularly difficult. Nevertheless, this case arises frequently in real queries. There are two cases that need to be differentiated. If the select

operator predicate limits the values of the join attribute, result size estimation is estimated by integration over the join domain limited to the range satisfying the selection predicate.

If the selection is performed on different attribute than the join attribute, co-moments between the domains (within one relation) and the multi-dimensional density function are required. The one-dimensional density function of the join domain can be derived and then be used to estimate the size of the join result. If all density functions are polynomials with known coefficients, deriving a density function from another of more dimensions is relatively easy and can be done symbolically. For the following equations, assume that the selection predicate is $L \leq x \leq U$ and that the join predicate is $y=z$ on domain A, with attributes x and y from one input relation and z from the other. The density function of the selection result is

$$f_{y \text{ after selection}}(y) = \int_{L}^{U} f_{x,y}(x,y)\, dx$$

The join result size can be calculated using the formula

$$\int_{A} \int_{L}^{U} f_{x,y}(x,a)\, dx\, f_{z}(a)\, da$$

If the join predicate is not an equality but an inequality (theta join) or if both equality and inequality clauses appear in the join predicate, density functions can also be used to estimate the result size.

Similar to selection results, it is possible to anticipate the density function of attributes in join results. We intend to explore this issue intensively, because it will be essential for reliable estimations in complex queries with deeply nested trees of operators.

6. Results Sizes of Projections and Aggregate Functions

The projection operator removes attributes from all its input tuples, leaving the attributes given in the *projection list.* Since relations are defined not to contain duplicate tuples, it is necessary to find and remove duplicates, except if the projection list contains a key, thus ensuring uniqueness.

Aggregates are functions, e.g. sum, count, or maximum, of the values of one attribute, e.g. salaries. Two forms of aggregates are distinguished in database systems, scalar aggregates and aggregate functions. Scalar aggregates result in a single value. This value can be determined separately from the main query and inserted as a constant into the modified query.

Aggregate functions classify the input tuples by a list of attributes. Following QUEL, we call this list the *by-list*. A frequently used example is the "sum of salaries by department". With respect to result cardinalities, aggregate functions can be considered a special form of projection. Instead of removing duplicates with equal values in the attributes of the projection list, aggregate functions combine (aggregate over) tuples with the same set of values in the attributes of the by-list.

In some respect, the problem of estimating the result size for projections is a dual of the problem of estimating the result size for a selection with an equality constraint. Basically, if we know the input size of a project operation and the number of equal values for each value, we can easily calculate the number of distinct values. Thus, we can use the principles outlined above for equality selections for projections and aggregate functions. The advantage of our method using density functions over current methods is that the result size can be estimated fairly accurately even if the operation follows another operator, e.g. a selection or a join, and we can estimate the distribution statistics fo the result.

7. Preliminary Results

In this section, we present some very preliminary results. In order to assess the practicality of the techniques, we developed a program that reproduces a given density function using random samples and moments. The density function to be reproduced is coded as a subroutine which can be changed without changes to the other parts of the program. The program operates in six steps. First, the integral of the original density is estimated numerically and normalized to be 1. Second, random numbers are chosen from a uniform distribution and mapped to reflect the density function to be reproduced. To be more precise, for the density

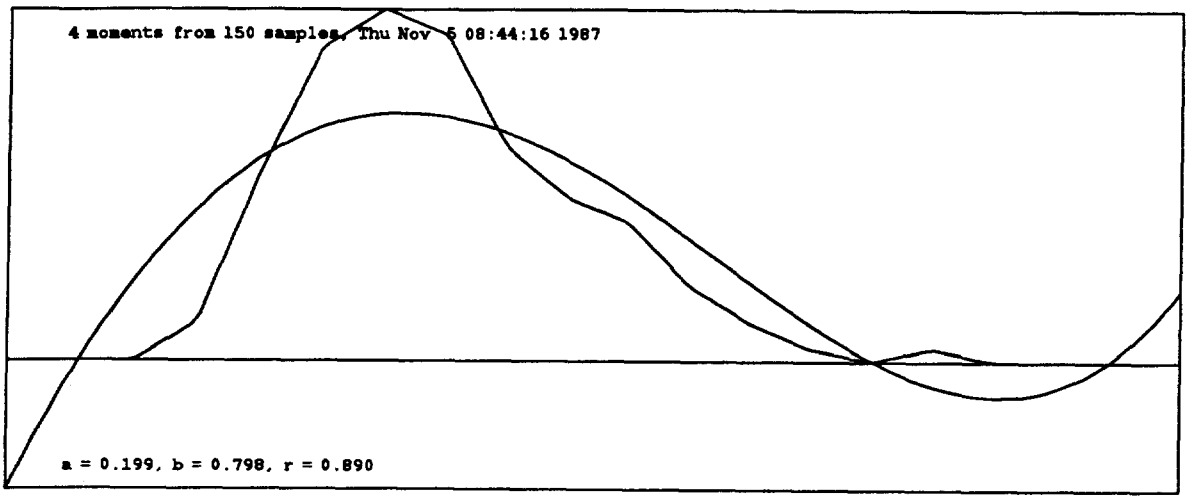
function f we map y_0 to x_0 such that

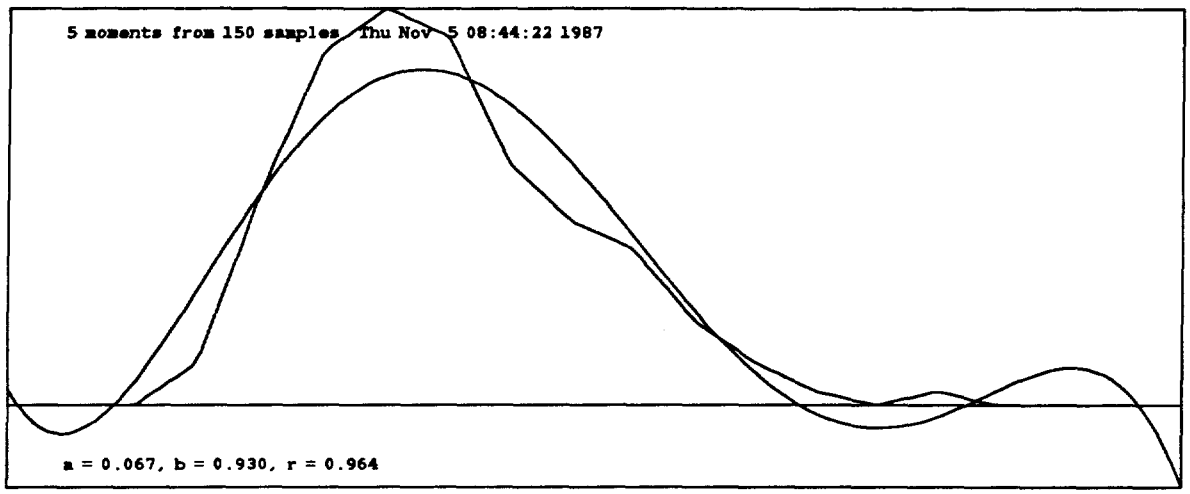
$$y_0 = \int_0^{x_0} f(x)\, dx.$$

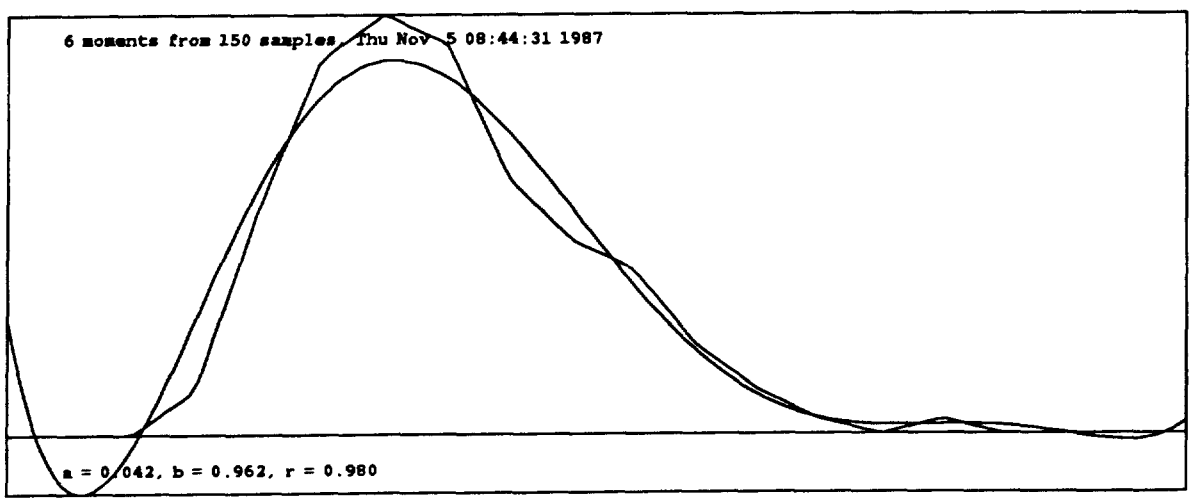
Third, the moments are gathered by adding powers of the mapped random numbers. Fourth, density functions in the form of polynomials are calculated by solving a system of linear equations, as described above. Fifth, the original density function and the calculated polynomial are evaluated for equidistant points over the range from minimum to maximum value[4], and linear regression coefficients and correlation coefficient r of the two sequences are calculated. Finally, the original density function and the calculated polynomial are plotted using these two sequences.

In the following graphs, we used a density function that we picked naively to reflect the salary distribution in an organization. Using 150 sample values (from 150 employees, so to speak), we calculated the moments. Using $M+1$ moments (M moments and the cardinality), we approximated the density function with a polynomial of degree M. The density functions for M = 3, 4, and 5 are shown. The polygon is the density function to be reproduced, while the smooth curve is the approximating polynomial. The x-axis ranges from 0 to 1. The y-axis is scaled such that the integral of the original density functions is 1. The horizontal line indicates the x-axis, i.e. $y=0$.

[4] In this preliminary program, the minimum is always 0 and the maximum is 1. Generalizing this range requires only a linear transformation which would not change the other calculations. Thus, this restriction is not significant. We evaluated the original and calculated density function at as many points as necessary to allow plotting seemingly smooth graphs, typically 150 points. The regression and correlation coefficients are affected only marginally by the chosen number of points.







Several observations can be made on these graphs. First, the approximations become increasingly more accurate as the number of moments and the degree of the polynomial

increase. This is apparent both visually and in the correlation coefficient r, printed in the legend of each graph. Second, the approximating density function takes values less than 0, i.e. the graph crosses the x-axis. This is an unavoidable result of Gibb's phenomenon (see, for example, [Hamming1977a, Oppenheim1983a]). Approximating a non-smooth curve with smooth functions always bears the risk of "overswings". We intent to explore the existing work on sharpening digital filters (see, for example, [Hamming1977a]) to determine how to reduce this problem. Third, the difference between the original and the approximating density function is largest close to the minimum and the maximum, i.e. the left and right end of the curves. We hope to eliminate most of this undesirable effect by using window functions (see, for example, [Hamming1977a, Oppenheim1983a]).

8. Finding and Updating Moments

There are essentially two ways to maintain the statistics used in query optimization, i.e. the moments of the attributes in a database. First, one can collect them periodically, and assume that they will not change significantly in the mean time. In order to assess the justification of this approach, the notion of significant change needs to be explored[5]. Second, the moments can be updated incrementally each time a tuple is modified, inserted, or deleted in the database. Moments are very well suited to incremental update. We will suggest some implementation mechanisms for each of these two methods in turn.

8.1. Periodic Update

Gathering the moments for the relations and attributes in a relational database requires scanning all files. We imagine that this can be done very fast using techniques like read-ahead. The processing load for each tuple are very low. For each attribute and each moment collected, one addition and one multiplication is required. We expect that, using a mainframe computer,

[5] We hope to include this issue in our research into the stability of access plans, mentioned in an earlier footnote.

this can be done at disk speed.

Randomly drawn samples have been reported in [Piatetsky-Shapiro1984a] for approximating inverted histograms. This technique is promises advantages only for very large files since it requires random access to both records and blocks. We will investigate whether randomly drawn samples are an appropriate method to estimate the moments. The preliminary results reported in Section 7 indicate that very small samples, in this case 150 values, give a reasonably close approximation of the underlying distribution.

8.2. Incremental Update

Most database systems update the statistics used in query optimization periodically. Even if the statistics are not exact in the meantime, they are sufficiently accurate considering the fact that the database query optimizer is based on untested assumptions and heuristics. This is a valid argument; we believe, however, the strongest argument is that maintaining correct statistics with each update inflicts too much overhead for database transactions. Using moments as the primary statistics, however, this argument may become invalid. In fact, we believe it will even be possible to maintain transaction consistent statistics which can easily merged with the statistics in the database catalogs when the transaction commits.

Consider how the moments change when a tuple is inserted, deleted, or modified in a database. Since moments are sums of attribute values raised to a power, they must be increased when a tuple is inserted, decreased when a tuple is deleted, and decreased according to the old amount and increased according to the new amount when a tuple is modified. In order to maintain transaction consistent moments, a transaction uses the moments from the database catalogs as a starting point and keeps track of the cumulated changes to the moments. Thus, within a transaction, queries can be optimized using exact statistics and density functions. If the transaction aborts, nothing needs to be done concerning the moments in the database catalogs. If the transaction commits, the cumulated changes (increases, decreases) initiated by the transaction can be applied to the database catalogs. Notice that the transaction cannot simply

overwrite the moments in the catalogs because another transaction may have updated the catalogs in the meantime. The limitation to updates only by increases and decreases allows to exploit high performance locking techniques, e.g. using the Escrow method [O'Neil1986a] modified to take advantage of the fact that illegal updates (overflow, underflow) are impossible.

We are not in the position yet to anticipate whether or not transaction consistent statistics are worth the effort. We feel, however, that moments can be updated as efficiently as the catalog entries for relation cardinalities, thus justifying a new look at the promise of transaction consistent statistics.

9. Summary and Conclusions

In this paper, we suggest to use statistical moments and density functions to estimate selectivities and result sizes of relational operations. The problem is of both theoretical and practical importance for database query optimization. Previous methods are based on assumptions that many real databases do not justify, namely the uniformity assumption and the independence assumption. Moments and density functions can be considered as a way to abandon both assumptions, thus allowing for more accurate estimates and better query optimization. The estimation procedures using moments and density functions are more complex than those currently used in database systems, but moments allow to determine a priori whether the effort is justified. It is important to note that moments can be calculated and maintained very efficiently even for very large databases.

The problem of estimating sizes of intermediate results has not received very much attention in database research, even though it is of substantial practical importance. We regard moments and density functions as a promising new approach to the problem.

References

Blasgen1977a.
M. Blasgen and K. Eswaran, "Storage and Access in Relational Databases," *IBM Systems Journal* **16**(4)(1977).

Cheney1980a.
W. Cheney and D. Kincaid, *Numerical Mathematics and Computing,* Brooks/Cole, Monterey, CA. (1980).

Christodoulakis1983a.
S. Christodoulakis, "Estimating Record Selectivities," *Information Systems* **8**(2) pp. 105-115 (1983).

Epstein1979a.
R. Epstein, "Techniques for Processing of Aggregates in Relational Database Systems," *UCB/ERL Memorandum,* (M79/8)University of California, (February 1979).

Graefe1987a.
G. Graefe, "The Stability of Query Evaluation Plans and Dynamic Query Evaluation Plans," *NSF Research Proposal,* Oregon Graduate Center, (1987).

Hamming1977a.
R.W. Hamming, *Digital Filters,* Prentice-Hall, Englewood Cliffs, NJ. (1977).

Jarke1984a.
M. Jarke and J. Koch, "Query Optimization in Database Systems," *ACM Computing Surveys* **16**(2) pp. 111-152 (June 1984).

Kooi1982a.
R.P. Kooi and D. Frankforth, "Query Optimization in Ingres," *Database Engineering* **5**(3) pp. 2-5 IEEE, (1982).

O'Neil1986a.
P.E. O'Neil, "The Escrow Transaction Method," *ACM Transactions on Database Systems* **11**(4) pp. 405-430 (December 1986).

Oppenheim1983a.
A.V. Oppenheim, A.S. Willsky, and I.T. Young, *Signals and Systems,* Prentice-Hall, Englewood Cliffs, NJ. (1983).

Piatetsky-Shapiro1984a.
G. Piatetsky-Shapiro and C. Connell, "Accurate Estimation of the Number of Tuples Satisfying a Condition," *Proceedings of the ACM SIGMOD Conference,* pp. 256-276 (June 1984).

Selinger1979a.
P. Griffiths Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, and T.G. Price, "Access Path Selection in a Relational Database Management System," *Proceedings of the ACM SIGMOD Conference,* pp. 23-34 (May-June 1979).

Stonebraker1976a.
M. Stonebraker, E. Wong, P. Kreps, and G.D. Held, "The Design and Implementation of INGRES," *ACM Transactions on Database Systems* **1**(3) pp. 189-222 (September 1976).

Wong1976a.
E. Wong and K. Youssefi, "Decomposition - A Strategy for Query Processing," *ACM Transactions on Database Systems* **1**(3) pp. 223-241 (September 1976).

Yang1985a.
D. Yang, "Expections Associated with Compound Selection and Join Operations," *Computer Science Technical Report,* (RM-85-02)University of Virginia, (July 1985).

Yao1979a.
S.B. Yao, "Optimization of Query Evaluation Algorithms," *ACM Transactions on Database Systems* **4**(2) pp. 133-155 (June 1979).

Zanden1986a.
B.T Vander Zanden, H.M. Taylor, and D. Bitton, "Estimating Block Accesses When

Attributes Are Correlated," *Proceeding of the Conference on Very Large Databases,* pp. 119-127 (August 1986).