Balancing Histogram Optimality and Practicality for Query Result Size Estimation

Yannis E. Ioannidis* Viswanath Poosala
{yannis,poosala}@cs.wisc.edu
Computer Sciences Department
University of Wisconsin
Madison, WI 53706

Abstract

Many current database systems use histograms to approximate the frequency distribution of values in the attributes of relations and based on them estimate query result sizes and access plan costs. In choosing among the various histograms, one has to balance between two conflicting goals: optimality, so that generated estimates have the least error, and practicality, so that histograms can be constructed and maintained efficiently. In this paper, we present both theoretical and experimental results on several issues related to this trade-off. Our overall conclusion is that the most effective approach is to focus on the class of histograms that accurately maintain the frequencies of a few attribute values and assume the uniform distribution for the rest, and choose for each relation the histogram in that class that is optimal for a self-join query.

1 Introduction

Query optimizers of relational database systems decide on the most efficient access plan for a given query based on a variety of statistics on the contents of the database relations that the system maintains. These statistics usually take the form of approximations of the distributions of data values in attributes of the relations and, hence, represent an inaccurate picture of the actual contents of the database. Since they are used to estimate the values of several parameters of interest to optimizers, e.g., intermediate result sizes, the validity of the optimizer's decisions may be affected [22]. Earlier work has shown that errors in query result size estimates may increase exponentially with the number of joins [10], which in conjunction with the increasing complexity of queries, demonstrates the critical importance of good-quality estimation.

Several techniques have been proposed in the literature to estimate query result sizes, most of them contained in the extensive survey by Mannino, Chu, and Sager [16] and elsewhere [4]. Those based on *sampling* primarily operate at run-time [7, 8, 15] and compute their estimates by collecting and possibly processing random samples of the data. Sampling is quite expensive and, therefore, its practicality is questionable, especially since it is performed mostly at run time and optimizers need query result size estimations frequently. Nevertheless, it often results in highly accurate estimates even in a high-update environment and avoids storing any statistical information in the database.

The techniques that do store information in the database may be roughly classified as *parametric* and *non-parametric* [4, 16]. In parametric techniques, the actual value distribution is approximated by a parameterized mathematical distribution. Although requiring very little overhead, this approach is typically inaccurate because real data does not usually follow any known distribution. Non-parametric techniques can be further subdivided into *algebraic* and *histogram*-based. In algebraic techniques, the actual value distribution is approximated with a polynomial, and thus problems similar to those of parametric techniques arise. A promising algebraic technique was recently proposed calling for adaptively approximating the distribution by a six-degree polynomial based on query feedbacks [1]. Its advantages are that there is very little overhead in maintaining the necessary information and that the approximation adapts well to updates. It suffers, however, in modeling distributions with many peak frequencies and gives inaccurate answers in data ranges that have been queried little. Probably the most common technique used in practice (e.g , DB2, Informix, Ingres) is maintaining *histograms*, where a histogram contains the number of tuples in a relation for each of several subsets (buckets) of values in an attribute. This is simple and relatively inexpensive, but requires effort to calculate the necessary number of buckets and also identify the attribute values that

*Partially supported by the National Science Foundation under Grants IRI-9113736 and IRI-9157368 (PYI Award) and by grants from DEC, IBM, HP, AT&T, and Informix.

SIGMOD ' 95,San Jose , CA USA
© 1995 ACM 0-89791-731-6/95/0005..$3.50

should be assigned to each bucket to achieve good estimates. Addressing some of these issues related to histograms is the topic of this paper.

Although histograms are used in many systems, their formal properties have not been studied extensively. To the best of our knowledge, earlier work deals with histograms in the context of single operations, primarily selection. Specifically, Piatetsky-Shapiro and Connell dealt with the effect of histograms on reducing the error for selection queries [21]. They studied two classes of histograms: in an *equi-width* histogram, the number of attribute values associated with each bucket is the same, in an *equi-depth* (or *equi-height*) histogram, the total number of tuples having the attribute values associated with each bucket is the same. Their main result showed that equi-width histograms have a much higher worst-case and average error for a variety of selection queries than equi-depth histograms. Muralikrishna and DeWitt [19] extended the above work for multidimensional histograms that are appropriate for multi-attribute selection queries. Several other researchers have dealt with "variable-width" histograms for selection queries, where the buckets are chosen based on various criteria [13, 18, 20]. The survey by Mannino, Chu, and Sager [16] contains various references to work in the area of statistics on choosing the appropriate number of buckets in a histogram for sufficient error reduction. That work deals primarily with selections as well. Histograms for single-join queries have been minimally studied and then again without emphasis on optimality [2, 14, 20].

Our on-going work is different from all the above in that it deals with arbitrarily large join and selection queries and focuses on identifying optimal or close-to-optimal histograms. In earlier efforts [9, 11], we studied individual tree equality-join queries for which the result size reaches some extreme, which essentially represent worst cases of error. Under the assumption that the frequency distributions of the relations of such a query can be examined collectively as a whole when constructing histograms, we showed that optimality for error reduction is achieved by the class of *serial* histograms, which we introduced.

Unfortunately, there is some difficulty in applying our earlier results in practical systems. Specifically, given an attribute of a relation, the optimal (serial) histogram for this attribute depends on the query in which the relation participates, the number of relations in the query, and their precise contents. This has two implications: first, to identify the optimal histogram (and verify that we are indeed dealing with the worst-case), the necessary information mentioned above must be derived, which essentially requires that the query be executed once; second, the optimality of a histogram for a relation's attribute is sensitive to changes in the query or the contents of the other query relations. In addition, although these results allow us to avoid the worst case, that case does not always arise in practice. Finally, even if all necessary frequency information was available for free, construction of the optimal serial histograms is expensive.

In this paper, motivated by the above issues, we investigate the trade-offs between histogram optimality and practicality. First, we show that if the result size of a query does not reach an extreme, serial histograms are not necessarily optimal, thus demonstrating the sensitivity of histogram optimality to changes in the database contents. Second, we prove that serial histograms *are* optimal on the average when the frequency distributions of the relations of a query are individually available but are examined in isolation, which is the most common situation in practice. Most importantly, the optimal histogram on a join attribute of a query relation is proved to be independent of the rest of the query and of the contents of the remaining query relations, and is equal to the optimal histogram for the query that joins the relation with itself on that attribute. Thus, optimal histograms can be identified independently for each relation, which is very critical for practical systems. Third, we study the subclass of serial histograms that accurately maintain the frequencies of some attribute values and assume the uniform distribution for the rest, and compare the optimal histogram in that class to the overall optimal (serial) histogram. We show that the histograms in this subclass are effective in error reduction and demonstrate that the optimal such histogram can be constructed much more efficiently than the optimal serial histogram. Fourth, we provide error formulas that can be used by the database system to advise administrators on the number of buckets required by a histogram for tolerable errors. Finally, we present a set of experimental results that show how the error in query result size estimates is affected by the compromises that we made for practicality, i.e., using an easy-to-construct but suboptimal class of histograms and concentrating on the average case. These results demonstrate the effectiveness of the overall approach.

2 Problem Formulation

2.1 Matrix Definitions

An $(M \times N)$-*matrix* $\underline{A}$ whose entries are $a_{kl}, 1 \leq k \leq M, 1 \leq l \leq N$, is denoted by $\underline{A} = (a_{kl})$. The results in this paper hold for matrices with non-negative real entries. For database applications, all entries will be non-negative integers. We occasionally use the terms *horizontal vector* and *vertical vector* for matrices with $M = 1$ and $N = 1$, respectively.

2.2 Problem Formulation

The focus of this paper is on tree function-free equality-join (and equality-selection) queries. We often omit

all qualifications and simply use 'query'. Without loss of generality, we assume that joins between relations are on individual attributes. For example, if R_0, R_1 are relation names and a, b are attributes of both, we do not deal with queries whose qualifications contain $(R_0.a = R_1.a$ **and** $R_0.b = R_1.b)$. Also without loss of generality, we only deal with *chain* join queries, i.e., ones whose qualification is of the generic form

$$Q := (R_0.a_1 = R_1.a_1 \quad \textbf{and} \quad R_1.a_2 = R_2.a_2 \ \textbf{and} \ \ldots \ \textbf{and} \quad R_{N-1}.a_N = R_N.a_N),$$

where $R_0, \ldots, R_N$ are relations and $a_1, \ldots, a_N$ are appropriate attributes. Generalizing the results presented in this paper to arbitrary tree queries is straightforward. The required mathematical machinery becomes hairier (tensors must be used) but its essence remains unchanged. Finally, it is well known that a selection can be considered as a special case of a join. For example, if R_0 is singleton and $a_1 = c$ is its sole tuple, then Q is equivalent to a query that contains the selection $R_1.a_1 = c$. In fact, a multi-tuple relation R_0 can be used to represent selections of the form $(R_1.a_1 = c_1$ **or** $R_1.a_1 = c_2$ **or** $\ldots$ **or** $R_1.a_1 = c_m)$. Hence, although in the rest of the paper we only mention joins, our results are applicable to queries with this type of selections as well.

Consider the above query Q and let $\mathcal{D}_j = \{d_{ij} | 1 \leq i \leq M_j$ for some integer $M_j\}$ be the (finite) *domain* of attribute $a_j, 1 \leq j \leq N$. (Whether $\mathcal{D}_j$ contains all the potential values that could appear in a_j or only those that actually appear at some point does not affect the results below.) Also, for convenience, let $M_0 = M_{N+1} = 1$. Note that $i < k$ does not imply $d_{ij} < d_{kj}$, i.e., the numbering of attribute values is arbitrary and does not reflect some natural ordering of them. The *frequency matrix* $\underline{T}_j = (t_{kl})$ of relation $R_j, 0 \leq j \leq N$, is defined as an $(M_j \times M_{j+1})$-matrix, whose entry t_{kl} is the *frequency* of the pair $< d_{kj}, d_{l(j+1)} >$ in the attributes a_j, a_{j+1} of R_j. Note that the frequency matrices of R_0 and R_N are a horizontal and a vertical vector, respectively. Occasionally, it is useful to treat all frequencies in $\underline{T}_j$ as a collection, ignoring the attribute value with which each frequency is associated. That collection is called the *frequency set* of R_j and may contain duplicates.

Example 2.1 A common claim is that, in many attributes in real databases, there are few domain values with high frequencies and many with low frequencies [3, 6]. Hence, for most examples in this paper, frequency distributions follow the Zipf distribution [24], which has exactly the above property. For a relation size T and domain size M, the frequencies generated by the Zipf distribution are

$$t_i = T \frac{1/i^z}{\sum_{i=1}^{M} 1/i^z} \text{ for all } 1 \leq i \leq M. \qquad (1)$$

Figure 1 is a graphical representation of (1) for $T = 1000$, M=100, and $z = 0, 0.02, \ldots, 0.1$, where the x-axis represents i, the rank of the attribute value with respect to its associated frequency in descending order. The skew of the Zipf distribution is a monotonically

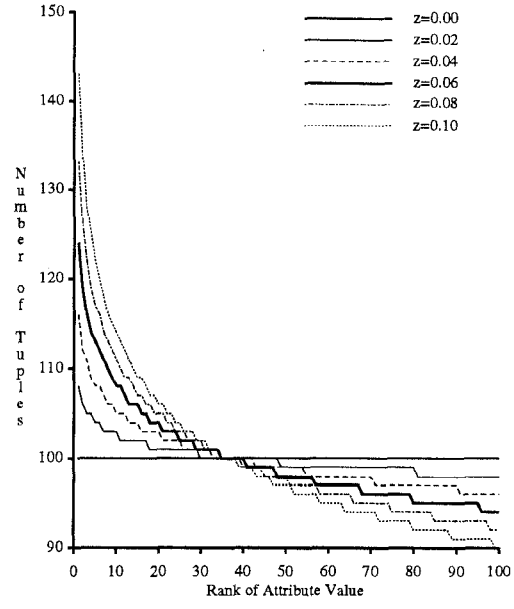

Figure 1: Zipf frequency distribution.

increasing function of the z parameter, starting from $z = 0$, which is the uniform distribution. □

Abstractly, in a database system, a frequency set is represented as a single-column table (we use the word 'table' not to be confused with query relations). Similarly, a D-dimensional frequency matrix is represented by a $(D+1)$-column table, with D *domain columns* for the cross product of the domains of its D dimensions and one *frequency column* for the frequency. We often need to collect in one structure the frequencies of the corresponding attribute values from all query relations. This is achieved by applying the query joins on the domain columns of the tables representing the frequency matrices, and generating a $(2N+1)$-column table with all the original columns kept in the result, i.e., N domain columns and $N+1$ frequency columns. In a slight misuse of terminology, we say that the result table represents the *joint-frequency matrix* of the query relations.

Theorem 2.1 [1] The size S of the result relation of query Q is equal to the product of the frequency matrices of its relations:

$$S = \underline{T}_0 \, \underline{T}_1 \, \cdots \, \underline{T}_N. \qquad (2)$$

[1] All results in this paper are given without proof due to lack of space. The full details can be found elsewhere [12].

Example 2.2 Consider the following query

$$Q := (R_0.a_1 = R_1.a_1 \textbf{ and } R_1.a_2 = R_2.a_2),$$

whose join attribute values have the frequencies below:

Rel	Domain	Frequencies
R_0	$\{v_1, v_2\}$	$v_1 \rightarrow 20$, $v_2 \rightarrow 15$
R_1	$\{v_1, v_2\} \times$	$< v_1, u_1 > \rightarrow 25$, $< v_2, u_1 > \rightarrow 12$,
	$\{u_1, u_2, u_3\}$	$< v_1, u_2 > \rightarrow 10$, $< v_2, u_2 > \rightarrow 4$,
		$< v_1, u_3 > \rightarrow 6$, $< v_2, u_3 > \rightarrow 3$
R_2	$\{u_1, u_2, u_3\}$	$u_1 \rightarrow 21$, $u_2 \rightarrow 16$, $u_3 \rightarrow 5$

The corresponding frequency matrices are

$$\underline{T}_0 = (\,20\;15\,),\ \underline{T}_1 = \begin{pmatrix} 25 & 10 & 6 \\ 12 & 4 & 3 \end{pmatrix},\ \underline{T}_2 = \begin{pmatrix} 21 \\ 16 \\ 5 \end{pmatrix}.$$

One can easily verify that the size of the result of Q is equal to $S = \underline{T}_0\underline{T}_1\underline{T}_2 = 19,265$. Some of the quintuples in their joint-frequency matrix are $< v_1, u_1, 20, 25, 21 >$, $< v_1, u_2, 20, 10, 16 >$, and $< v_2, u_3, 15, 3, 5 >$.

As an example of a query with a selection, consider

$$Q' := (R_0.a_1 = R_1.a_1 \textbf{ and } (R_1.a_2 = u_1 \textbf{ or } R_1.a_2 = u_3)).$$

The size of the result of Q' can be computed as before but by using the transpose (i.e., vertical) vector of (1 0 1) instead of $\underline{T}_2$, the 1's indicating that u_1 and u_3 are the values selected. □

2.3 Histograms

Among commercial systems, maintaining *histograms* is a very common approach to approximating frequency matrices. In what follows, we discuss histograms for two-dimensional matrices; histograms for matrices of other dimensions are defined similarly. In a histogram on attributes a_j, a_{j+1} of relation $R_j, 0 < j < N$, the set $\mathcal{D}_j \times \mathcal{D}_{j+1}$ is partitioned into *buckets*, and a uniform distribution is assumed within each bucket. That is, for any bucket b in the histogram, if $< d_{kj}, d_{l(j+1)} > \in b$ then t_{kl} is approximated by the integer closest to $\sum_{< d_{mj}, d_{n(j+1)} > \in b} t_{mn}/|b|$. The approximate frequency matrix captured by a histogram is called a *histogram matrix*. This is typically stored in some compact form in the catalogs of the database system, and the optimizer uses the average frequency corresponding to a bucket to approximate the frequencies of all domain values belonging to it. A histogram whose matrix has all its entries equal is called *trivial* and corresponds to making the uniform distribution assumption. Note that any arbitrary subset of $\mathcal{D}_j \times \mathcal{D}_{j+1}$ may form a bucket, e.g., bucket $\{< d_{1j}, d_{4(j+1)} >, < d_{7j}, d_{2(j+1)} >\}$. Whenever no confusion arises, we may describe a histogram as placing not combinations of domain values in its buckets but frequencies.

After any update to a relation, the corresponding histogram matrix may need to be updated as well. Otherwise, delaying the propagation of database updates to the histogram may introduce additional errors. Appropriate schedules of database update propagation to histograms are an issue that is beyond the scope of this paper. Moreover, any proposals in that direction do not affect the results presented here, so this issue is not discussed any further.

Example 2.3 To illustrate the above definition of histograms, consider the following relation schema: WorksFor(*ename,dname,*year). The attributes in italics form the key to the relation, and represent employee and department names such that the employee is working in the department. The 'year' attribute represents the year the employee started working at the department. We focus on the combination of 'dname' and 'year' attributes, and assume for simplicity that there are four different departments (toy, jewelry, shoe, and candy) and five different years (1990 through 1994). The frequency matrix of the relation is shown in Figure 2(a), where 'dname' is used for the rows and 'year' is used for the columns of the matrix, and the corresponding values are arranged in the order specified above. An example histogram matrix with two buckets is shown in Figures 2(b) and 2(c). In the former, we show the original matrix with an indication of which attribute value pairs (or equivalently, frequencies) are placed in which bucket. In the latter, we show the actual histogram matrix that is the result of averaging the frequencies in each bucket. Another example histogram matrix with two buckets is shown in Figures 2(d) and 2(e), again depicted in the two ways discussed for the first histogram. □

We now define a very important class of histograms.

Definition 2.1 Consider relation $R_j, 0 \leq j \leq N$, with a frequency matrix $\underline{T}_j$. A histogram for relation R_j is *serial*, if for all pairs of buckets b_1, b_2, either $\forall < d_{kj}, d_{l(j+1)} > \in b_1, < d_{mj}, d_{n(j+1)} > \in b_2$, the inequality $t_{kl} \geq t_{mn}$ holds, or $\forall < d_{kj}, d_{l(j+1)} > \in b_1, < d_{mj}, d_{n(j+1)} > \in b_2$, the inequality $t_{kl} \leq t_{mn}$ holds.

Note that the buckets of a serial histogram group frequencies that are close to each other with no interleaving. For example, the histogram of Figures 2(d)-(e) is serial, while that of Figures 2(b)-(c) is not.

A histogram bucket whose domain values are associated with equal frequencies is called *univalued*. Otherwise, it is called *multivalued*. Univalued buckets characterize the following important classes of histograms.

Definition 2.2 A histogram with $\beta - 1$ univalued buckets and one multivalued bucket is called *biased*. If the univalued buckets correspond to the domain values

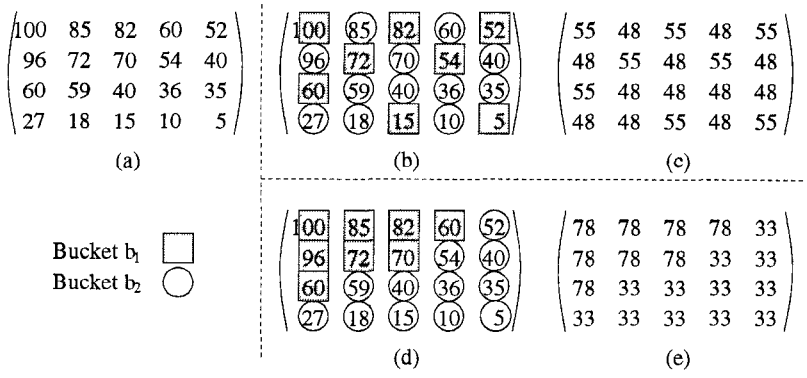

Figure 2: Frequency matrix and two histogram matrices on WorksFor.

with the β_1 highest and the β_2 lowest frequencies, where $\beta - 1 = \beta_1 + \beta_2$, then it is called *end-biased.*

Note that end-biased histograms are serial.

2.4 Histogram Optimality

The accuracy of estimates of query result sizes obtained through histograms depends heavily on how the domain values of the attribute(s) concerned have been grouped into buckets [11, 21]. Ideally, for any given query and database, one would always like to use the histograms on the join attribute(s) of each relation that return the closest approximation to the result size of the specific query applied on the specific database. Several practical considerations, however, make this ideal unachievable.

First, taking into account the precise contents of the database is usually very expensive and results in histograms that are very sensitive to database changes. Therefore, it is important to use restricted forms of information about the database contents and obtain a histogram that is best on the average, for all possible database contents that are consistent with the available information. Our study deals with histogram optimality when the joint-frequency matrix of all query relations is known (full knowledge) and when only their frequency sets are known (which we consider to be the minimum required knowledge and is easily obtained in practice).

Second, taking into account the query in which relations participate results in histograms that may be very poor for other queries. Our study does focus on individual queries but presents results that show that, when only frequency sets are known, the best histogram is *independent* of the query.

Third, in all cases, identifying the optimal histograms requires algorithms that are exponential in cost. Moreover, efficient access to arbitrary histograms requires advanced data structures, which incur nontrivial space and maintenance overhead. Therefore, it is crucial to identify not the best overall histogram but the best one within a restricted but more efficient class. Our study deals with both arbitrary and biased histograms, since the latter have much lower construction and maintenance overhead.

The following section deals with histogram optimality under various amounts of available information, addressing the first two issues above. The subsequent section deals with differences in histogram construction and maintenance cost, addressing the third issue above.

3 Histogram Optimality under Varying Knowledge

3.1 Knowledge of Joint-Frequency Matrix

When the joint-frequency matrix of all query relations is available, optimal histograms are defined as follows.

Definition 3.1 Consider a query Q on relations $R_j, 0 \leq j \leq N$, whose result size is S, as determined by the frequency matrices of the relations. For each relation R_j, let $\mathcal{H}_j$ be a collection of histograms of interest. The $(N+1)$-tuple $< H_j >$, where $H_j \in \mathcal{H}_j, 0 \leq j \leq N$, is *optimal* for Q within $< \mathcal{H}_j >$, if it minimizes $|S - S'|$, where S' is the approximate query result size determined by any such histogram tuple.

Note that optimality is defined per query and per collection of frequency matrices, and for the histograms of all relations together.

In our previous work [9, 11], we mostly concentrated on the case where the joint-frequency matrix of the query relations is known, and dealt with histogram optimality when the query result size reaches some extreme. Below, we summarize the main results that we have obtained earlier for the case where that size is maximized, because they are important for the results presented in this paper. We then provide some new insights for the general (non-extreme) case.

We have investigated histogram optimality within the class of general histograms $\mathcal{H}$, and used $\mathcal{H}_\beta$ to refer to its subclass that contains only histograms with β buckets. Using results from the mathematical theory

of majorization [17] and extending them to matrices of arbitrary dimension, we have derived the following general theorem, which establishes the importance of serial histograms.

Theorem 3.1 [9] Consider a query Q on relations R_j, $0 \leq j \leq N$, and let B_j be their frequency sets. Assume that, for each $0 \leq j \leq N$, the elements of B_j are arranged in the frequency matrix $\underline{T}_j$ so that the result size of Q is maximized. Consider an $(N+1)$-tuple of histogram sizes $< \beta_j >$. There exists an optimal histogram tuple for Q within $< \mathcal{H}_{\beta_j} >$ where all histograms in it are serial.

Corollary 3.1 For the query in the above theorem, the optimal biased histogram for any relation is end-biased.

Histograms are usually constructed in such a way that each bucket stores attribute values that belong in a certain range in the natural total order of the attribute domain. The important implication of Theorem 3.1 is that this traditional approach may be far from optimal. (Examples are shown in Section 5.) Histograms should be constructed so that attribute values are grouped in buckets based on proximity in their corresponding frequencies (serial histograms) and not in their actual values. This is a significant difference, since the two orderings may be completely unrelated.

Identifying which of the many serial histograms is optimal in each case is not straightforward. Here we focus on 2-way join queries where a relation is joined with itself (self-joins), which is a case where the query result size is maximized [17]. For a self-join, the optimal histogram is the same for both instances of the joined relation [11] and may be identified by exhaustively looking at all possible bucketizations of the relation's frequencies and choosing the one minimizing the error.

Before presenting the necessary size and error formulas, we introduce some notation regarding a serial histogram:

- b_i The i-th bucket in the histogram, $1 \leq i \leq \beta$ (numbering is in no particular order).
- T_i The sum of the frequencies in bucket b_i.
- p_i The number of frequencies in bucket b_i.
- V_i The variance of the frequencies in bucket b_i.

Proposition 3.1 Consider a query Q joining a relation R with itself The approximate size of Q corresponding to a serial histogram of R with β buckets is equal to

$$S' = \sum_{i=1}^{\beta} \frac{T_i^2}{p_i},$$

while the error in the approximation is equal to

$$S - S' = \sum_{i=1}^{\beta} p_i \, V_i. \tag{3}$$

Since biased histograms are serial, the above formulas apply to them as well. In addition to its usefulness in algorithms identifying optimal serial and end-biased histograms, the above proposition has other applications also. By applying the error formula to histograms of various numbers of buckets, administrators can determine the minimum number of buckets required for tolerable errors. For example, when applied to distributions that are close to uniform, the value returned will be close to zero independent of the number of buckets, which would indicate that one or two buckets will suffice. Also, the above proposition provides some intuition to Theorem 3.1 and to why serial histograms are optimal. Intuitively, serial histograms group similar frequencies together, thereby reducing the variances V_i of frequencies in the buckets. Based on the formula for $S - S'$ above, this essentially reduces the estimation error.

Theorem 3.1 is inapplicable to arbitrary non-extreme cases, and in fact, any formal results are unlikely to exist in general. Nevertheless, given the frequency sets of two relations, for most corresponding frequency matrices (i.e., arrangements of the frequency sets), the associated optimal histograms do turn out to be serial We have experimented with various Zipf distributions and biased histograms for the relations of a 2-way join query. In approximately 90% of all arrangements, the optimal histogram pair places the frequencies of the same domain values in the univalued buckets and has at least one of the two histograms be end-biased (i.e., serial). Also, in about 20% of all arrangements, both histograms are end-biased. The insights gained from the above results and experimental evidence have guided some of the work that is presented in the next subsection, where less information about the database contents is assumed.

3.2 Knowledge of Frequency Sets Alone

In this subsection, we investigate histogram optimality when the frequency matrices of individual relations of a query are available but their joint-frequency matrix is not, i.e., when essentially only the frequency sets are known. This is the typical scenario in real systems, where statistics are collected independently for each relation, without any cross references. Using this knowledge, the goal is to identify optimal histograms for the average case, i.e., taking into account all possible permutations of the frequency sets of the query relations over their corresponding join domains.

We first introduce some formal notation. Assume that for each relation R_j, $0 \leq j \leq N$, in query Q, its frequency set B_j together with a partitioning of B_j into buckets are given. Let B'_j be the approximate frequency set generated from B_j by replacing each frequency by the average of the frequencies that belong to the same bucket in the given partitioning. For each relation $R_j, 0 \leq j \leq N$, consider all possible arrangements of the elements of B_j in the frequency matrix $\underline{T}_j$ that respect any functional dependencies that may hold in R_j, $0 < j < N$, and the same arrangements of the corresponding elements of B'_j. Each combination of arrangements in all matrices corresponds to a value for the actual query result size S and its approximation S'.

One could define optimal histograms as those that minimize the expected value of the difference $S - S'$, denoted by $E[S - S']$. Unfortunately, the following theorem from our earlier work shows that all histograms are equivalent in that respect.

Theorem 3.2 [9] Consider a query Q on relations R_j, $0 \leq j \leq N$, and an $(N+1)$-vector $< B_j >$ of frequency sets together with partitionings of them. If $E[S - S']$ is defined as above, then $E[S - S'] = 0$

Note that Theorem 3.2 deals with arbitrary histograms, not only serial ones. It implies that all histograms are accurate in their approximation of the expected value of the query result size. Hence, this quantity cannot be used to define optimal histograms.

We define optimal histograms as those that minimize the expected value of the square of the difference $S-S'$, denoted by $E[(S - S')^2]$. To distinguish that form of optimality from the one of Definition 3.1, we use the term *v-optimal*. Note that, since $E[S - S'] = 0$, this error measure $E[(S-S')^2]$ is also the variance of $(S-S')$ (the reason for the prefix v in v-optimal). Hence, v-optimal histograms could have been defined as those that minimize the variance of the difference between the actual and approximate sizes.

Definition 3.2 Consider a query Q on relations $R_j, 0 \leq j \leq N$, whose corresponding frequency sets are $B_j, 0 \leq j \leq N$. For each relation R_j, let $\mathcal{H}_j$ be a collection of histograms of interest. The $(N+1)$-tuple $< H_j >$, where $H_j \in \mathcal{H}_j, 0 \leq j \leq N$, is *v-optimal* for Q within $< \mathcal{H}_j >$, if it minimizes $E[(S - S')^2]$.

Note that, as in the case where the query result size is maximized, optimality is defined per query and per collection of frequency sets, and for the histograms of all relations together. The following general theorem identifies the v-optimal histogram for any query

Theorem 3.3 Consider a query Q on relations R_j, $0 \leq j \leq N$, and let B_j, $0 \leq j \leq N$, be their frequency sets. Consider an $(N+1)$-tuple of histogram sizes $< \beta_j >$. Let H_j, $0 \leq j \leq N$, be the optimal histogram within $\mathcal{H}_{\beta_j}$ for joining R_j with itself on its attributes that are part of Q. Then, the histogram tuple $< H_j >$ is v-optimal for Q within $< \mathcal{H}_{\beta_j} >$.

There are several important implications of Theorem 3.3.

1. The 2-way join of R_j, $0 \leq j \leq N$, with itself represents a case where the joint-frequency vector of the two joined relations is known and the query result size is maximized (self-join). Therefore, by Theorem 3.1, the v-optimal histograms are serial.

2. The v-optimal (serial) histogram for R_j, $0 \leq j \leq N$, is independent of the query and of the contents (i.e., the frequency set) of any other relation. It can, therefore, be identified by focusing on the frequency set of the relation alone. It will be v-optimal for any query where R_j is joined on the same attribute(s) with any relation of any contents.

3. Proposition 3.1 can be applied to precisely identify the v-optimal (serial) histogram of a relation.

Theorem 3.3 is a significant step towards practical histograms. It shows that v-optimality is a local property of each relation and can be obtained by looking into the individual frequency sets alone.

3.3 Comparison of Information Collection Costs

In the previous subsections, we have examined histogram optimality under two different amounts of available information. We now compare the costs incurred when obtaining this information.

For a given query Q joining relations R_0 and R_1 on attribute a_1, the joint-frequency matrix can be obtained by Algorithm JointMatrix below. (The algorithm can be easily extended for more joins; it is only presented here for a 2-way join for simplicity.)

*Construction (**JointMatrix**)*: First, the frequencies of the domain values of attribute a_1 in R_0 and R_1 are computed. This can be achieved in a single scan of each relation using a hash table to access the frequency counter corresponding to each data value. Next, these two lists of <attribute, frequency> pairs are joined on the attribute value to give the joint-frequency matrix.

This algorithm is quite expensive because of the join operation. The problem is simpler when we want to obtain the the frequency sets alone, which are given by JointMatrix immediately before the join step. The simpler algorithm is called Matrix and will be very efficient if an index exists on the attribute(s) of interest. Otherwise, even Matrix may become quite expensive, especially when the number of different values in the attributes of interest is very high, which is particularly possible when collecting statistics for multiple attributes of a relation. In that case, there may not be sufficient

space for the frequency tables in primary memory, which would force some disk I/Os. Nevertheless since statistics collection is an infrequent operation, this is usually acceptable We later discuss some efficient heuristic algorithms that are effective for certain common cases and require very little working space.

4 Construction and Storage Efficiency of Optimal Histograms

The results in the previous section addressed two of the three issues raised in Section 2.4 regarding the practicality of histograms. First, optimal histograms have been identified when only knowledge of frequency sets is available, and second, these histograms have been shown to be query independent. In this section, we address the last issue, dealing with the efficiency of construction and maintenance of histograms in various classes. Specifically, as suggested by the analysis of the previous section, we focus on the v-optimal serial and end-biased histograms and demonstrate that very efficient techniques exist for the latter while dealing with general serial histograms is very costly. Algorithm Matrix, which computes the individual frequency matrices of a query's relations, is in general a necessary first step in constructing any histogram, and is therefore ignored in the histogram comparison.

4.1 Serial Histograms

*Construction (**V-OptHist**)*: In conjunction with Proposition 3.1, Theorem 3.3 leads to the following algorithm V-OptHist for identifying the v-optimal histogram with β buckets of relation R with a given frequency set B: B is sorted and then partitioned into β contiguous sets in all possible ways. Each partitioning corresponds to a unique serial histogram with the contiguous sets as its buckets. The error corresponding to each histogram is computed using formula (3), and the histogram with the minimum error is returned as optimal.

Theorem 4.1 Given the frequency set B of a relation R and an integer $\beta > 1$, Algorithm V-OptHist finds the (serial) histogram that is v-optimal within $\mathcal{H}_\beta$ for any query of R. If the cardinality of B is M, the algorithm runs in $O(MlogM + (M-1)^{\beta-1})$ time.

Storage and Maintenance: Since there is usually no order-correlation between attribute values and their frequencies, for each bucket in the histogram, we need to store the average of all frequencies in it and a list of the attribute values mapped to the bucket. Efficient accessing of this information requires a multi-dimensional index, something too expensive for a catalog structure.

In storing a general serial histogram, one may save some space by not storing the attribute values associated with its largest bucket and only storing their approximate frequency in a special form. Not finding a valid attribute value among those explicitly stored implies that it belongs to the missing bucket and has that special frequency. Clearly, the larger that bucket is, the more space-efficient and realistic a histogram becomes. As an extreme case, consider a 1-bucket histogram (uniform approximation), where only the average of all frequencies is stored but not the attributes values, since all of them are associated with that average. As will be discussed below, end-biased histograms are another realistic special case.

4.2 End-biased histograms

As demonstrated above, except for very small domain sizes, there are two major problems with general serial histograms: the exponential complexity of Algorithm V-OptHist and the storage overhead. Because of these, we turn our attention to biased histograms, which suffer from no such problem.

*Construction (**V-OptBiasHist**)*: Since the serial biased histograms are end-biased, Theorem 3.3 implies that the v-optimal biased histogram for a relation R is end-biased and is the same histogram that is optimal for joining R with itself. The algorithm V-OptBiasHist to identify the v-optimal end-biased histogram is similar to V-OptHist, but simply enumerates all the end-biased histograms instead of all serial histograms. Fortunately, the number of end-biased histograms is less than the number of frequencies (M). They can be efficiently generated by using a heap tree to pick the highest and lowest $\beta - 1$ frequencies and enumerating the combinations of highest and lowest frequencies placed in univalued buckets. Since the univalued buckets have zero variance, the v-optimal end-biased histogram is the one whose multivalued bucket has the least variance (formula (3)).

Theorem 4.2 Given the frequency set B of a relation R and an integer $\beta > 1$, Algorithm V-OptBiasHist finds the v-optimal end-biased histogram with β buckets for any query of R. If the cardinality of B is M, then the algorithm runs in $O(M + (\beta - 1)logM)$ time.

Theorem 4.2 shows that the optimal end-biased histogram may be found very efficiently, in time that is almost linear. The difference with the algorithm for finding the optimal serial histogram is significant.

Certainly the cost of V-OptBiasHist is much less than the cost of Matrix, which constructs the frequency set of a relation. As mentioned above, Matrix may need to perform some disk I/Os when there are many distinct attribute values. The two algorithms may be combined in an efficient alternative when only high frequencies are picked for univalued buckets in the end-biased histogram. In that case, sampling can be used to identify the $\beta - 1$ highest frequencies, which is an

extremely fast operation, requiring constant amount of very small space. Something similar is done in DB2/MVS in order to identify the 10 highest frequencies in each attribute, which it maintains in its catalogs [23]. This approach will not work when the distribution has relatively many high frequencies and few small ones, in which case low frequencies will be chosen for univalued buckets, because there is no known efficient technique to identify the lowest frequencies in a distribution. Such distributions, however, are quite rare in practice (they are, in some sense, the reverse of Zipf distributions), so we expect that most often the efficient combination of V-OptBiasHist and Matrix would be applicable.

Storage and Maintenance: As mentioned above, end-biased histograms are a special case of serial histograms that require little storage, since most of the attribute values belong in a single bucket and do not have to be stored explicitly In addition, since the number of univalued buckets (β-1) tends to be quite small in practice ($\leq$ 20), binary or sequential search of the buckets will be sufficient to access the information efficiently. Note that many commercial database systems already contain catalogs intended to store similar statistics, e.g., the `SYSIBM.SYSCOLDIST` and `SYSIBM.SYSCOLUMNS` catalogs of DB2 [5].

4.3 Comparison of Histogram Construction Costs

Ignoring the cost of obtaining the frequencies of the attribute values (algorithm Matrix), the cost of histogram construction depends on the number of distinct values in the histogram attribute(s) and the number of desirable buckets. The following table illustrates the difference in construction cost between general serial and end-biased histograms (Theorems 4.1 and 4.2) by presenting actual timings collected from running algorithms V-OptHist and V-OptBiasHist on a DEC ALPHA machine, for varying cardinalities of frequency sets and numbers of buckets. For end-biased histograms, the numbers presented are for $\beta = 10$ buckets, and are very similar to what was observed for β ranging between 3 and 200. In contrast, for the serial histograms, the times increase drastically for a small increase in buckets from 3 to 5 and similarly for increases in the number of attribute values. Thus, the results presented are rather limited, but we believe they are sufficient to convey the construction cost difference between histograms.

Number of attribute values	Time taken (sec)		
	Serial		End-biased
	$\beta = 3$	$\beta = 5$	$\beta = 10$
100	0.18	128.60	0.00
1000	156.70		0.01
100K			0 10
1M			1.80

Table 1: Construction cost for optimal general serial and end-biased histograms

5 Effectiveness of Practical Histograms

Given a relation R and a (possibly singleton) set of its attributes A, let the *affordable histogram* with β buckets for A of R be the end-biased histogram with β buckets that is v-optimal, i.e., optimal for a self-join of R on A. The cumulative outcome of the previous two sections has been that affordable histograms are flexible (i.e., they work well on the average) and can be efficiently obtained in practice. Clearly, the price paid for this flexibility and efficiency is that affordable histograms are, in general, suboptimal. In this section, we investigate the magnitude of that price and present several experimental results that illustrate the effectiveness of affordable histograms in most real-life cases. Specifically, we first compare various types of histograms on self-join queries (which are important with respect to v-optimality) and study how errors are affected as different parameters vary (β, M, frequency distribution skew). We then compare the errors generated by the v-optimal end-biased and serial histograms for various queries on relations of different frequency distributions, thus investigating the effect of using reduced information (i.e., frequency sets).

We use five types of histograms: trivial, optimal serial, optimal end-biased, equi-width and equi-depth histograms, with the number of buckets β ranging from 1 to 30. In most experiments, the frequency sets of all relations follow the Zipf distribution, since it presumably reflects reality [3, 6, 24]. Its z parameter takes values in the range [0.01 – 3.0], which allows experimentation with varying degrees of skewness. The size M of the join domains ranges from 10 to 100.

5.1 Effect of Histograms Type

In this subsection, we compare the error generated by the five types of histograms mentioned above for self-joins, due to the importance of these queries for v-optimality. To correctly model the equi-depth and equi-width histograms, we assume no correlation between the natural ordering of the domain values and the ordering of their frequencies.

5.1.1 Synthetic Data

As mentioned above, the three parameters of interest are β, M, and z. The relation size (parameter T in (1)) has provably no effect on any result and was chosen arbitrarily to be 1000 tuples. Each one of

Figures 3, 4, and 5 shows the standard deviation of the error ($\sigma = \sqrt{E[(S-S')^2]}$, based on which v-optimality is essentially defined) generated by the five types of histograms as a function of one of these parameters while keeping the others constant. In almost all cases, the histograms can be ranked in the following order from lowest to highest error: optimal serial, optimal end-biased, equi-depth, equi-width, and trivial (uniformity assumption). (For the comparison of equi-depth with equi-width, we essentially verified the findings of Piatetsky-Shapiro and Connell [21].) The error of the optimal end-biased histogram is usually less than twice the error of the optimal serial histogram and much less than half the error of the equi-depth histogram. Hence, using serial histograms (the optimal serial or the optimal end-biased histogram) is crucial. Interestingly, the equi-width and the trivial histograms are almost always identical, the reason being that the association of domain values and frequencies was random and had no implication on their respective orderings

In Figure 3, β is varied, while $M = 100$ and $z = 1.0$. The optimal serial histogram is not shown for more that $\beta = 5$ buckets, because of the exponential time complexity of Algorithm V-OptHist. One can notice that, although the optimal serial and end-biased histograms always improve with additional buckets, the equi-width and especially the equi-depth histograms have nonmonotonic behavior. Although decreasing overall, the equi-depth histogram is quite unpredictable with many instances of increased errors for increased β. The trivial histogram is unaffected, since it always has a single bucket, and the equi-width histogram has very similar behavior as well. For the optimal serial and end-biased histograms, the improvement is more dramatic in the range of small number of buckets, while after a point additional buckets have little effect. From a practical standpoint, this is important since it implies that a small number of univalued buckets is enough to bring down the error to very low levels.

In Figure 4, M is varied, while $\beta = 5$ and $z = 1.0$. As expected, for a few values of M beyond $M = 5$, the error increases, since five buckets are not enough to accurately capture the distribution. Nevertheless, because the relation size remains always constant, as the domain size increases, the frequency distribution becomes increasingly more similar to uniform. Therefore, beyond a certain point the same number of buckets becomes more effective and the error decreases for all histograms.

In Figure 5, z is varied, while $\beta = 5$ and $M = 100$ The histograms are partitioned in two categories based on two very distinct behaviors. The equi-width and trivial histograms behave very poorly as the skew in the frequency distribution (or equivalently z) increases and fall quickly out of the chart. For the other three histograms, the diagram presents a maximum: beyond a certain value of z, further increases reduce the error. The reason is that all these histograms are essentially frequency-based. For high values of skew, there are few frequencies that are very high and, therefore, end up in univalued buckets, while the rest of the frequencies are very low and end up in a single multivalued bucket that generates a small error. Thus, as low skew is easy to handle because the choice of buckets is unimportant, similarly high skew is easy to handle because the choice of buckets is easy.

5.1.2 Real-life Data

We have also conducted experiments using frequency sets from a real-life database (performance measures of NBA players). Due to space limitations, we do not present the results of these experiments. Overall, however, these verified what was observed for the Zipf distribution, despite the wide variety of distributions exhibited by the data.

5.2 Effect of Amount of Available Information

Based on Theorem 3.3, the amount of available information used in histogram construction touches upon two of the issues raised in Section 2.4: use of joint-frequency matrices vs. frequency sets and use of individual queries vs. self-joins. The trade-offs that these issues introduce are examined in this section. We study arbitrary join queries on relations with different frequency sets.

In this experiment, we take the average of the results obtained for several permutations of the frequency sets of the relations involved. Since our goal is to study how affordable histograms behave in general, the y-axis of the graphs represents the mean relative error over all permutations, defined as $E[|S-S'|/S]$.

Specifically, the frequency sets of all relations follow the Zipf distribution, but z takes values in $\{0.0, 0.1, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0\}$. In addition, $M = 10$ for the two end relations of a query (which need one-dimensional matrices), but $M = 100$ for the remaining relations (which need (10×10) two-dimensional matrices). Three separate experiments are run for three classes of queries: low skew queries (where z is randomly chosen for each relation from the set $\{0.0, \ldots, 0.75\}$), mixed skew queries (where z is randomly chosen among all possible values), and high skew queries (where z is randomly chosen from the set $\{1.0, \ldots, 3.0\}$). For each class of queries and each number of joins, average errors are obtained over twenty permutations of the frequency sets generated. Given the size of domain of most relations, this experiment does not include any actually optimal histogram, but only compares the errors generated by the trivial, v-optimal serial, and v-optimal end-biased histograms with varying numbers of buckets. The errors generated in this experiment are shown

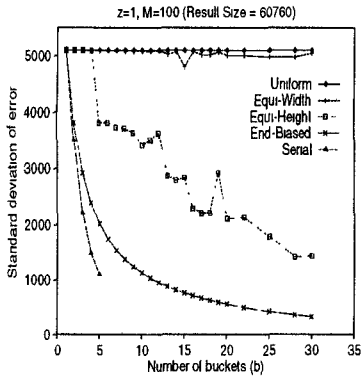

Figure 3: σ as a function of the number of buckets

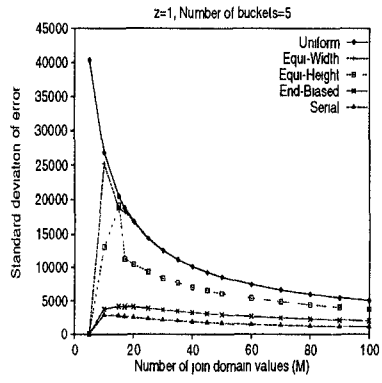

Figure 4: σ as a function of the join domain size.

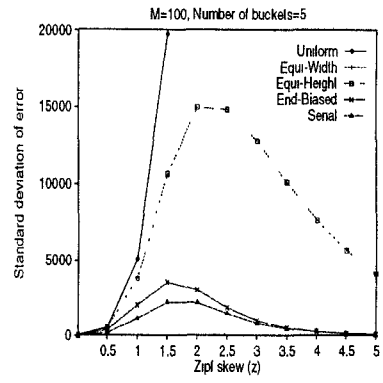

Figure 5: σ as a function of skew (z parameter of Zipf).

in Figure 6 as a function of the number of joins for $\beta = 5$ buckets, and in Figure 7 as a function of the number of buckets for five joins. The curves for the trivial histogram are not shown because, except for the low skew queries, they fall way outside of the charts. There are

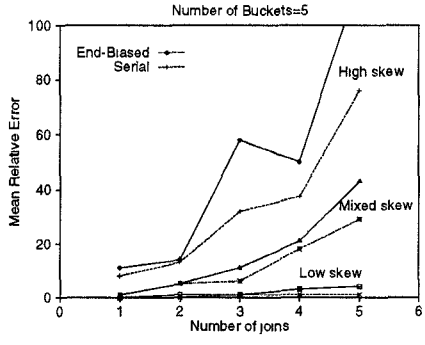

Figure 6: $E[|S - S'|/S]$ as a function of the number of joins.

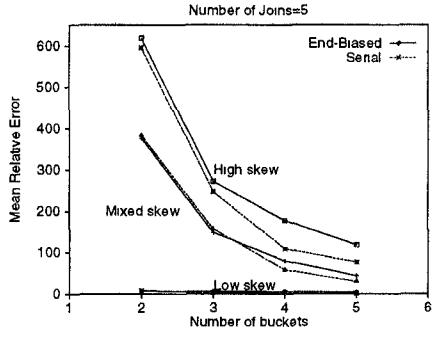

Figure 7: $E[|S - S'|/S]$ as a function of the number of buckets.

two main conclusions that may be drawn from these figures. First, as expected, the errors increase with the number of joins and decrease with the number of buckets. Interestingly, even with a small number of buckets ($\beta = 5$), the errors drop significantly to a tolerable level. Second, the v-optimal serial histograms is not always end-biased in the general case. Nevertheless, their average difference appears to be relatively small, and in fact the v-optimal serial histogram is sometimes worse than the end-biased histogram for arbitrary queries, as observed in mixed skew queries of five joins for two and three buckets. This implies again that, for most general queries, concentrating on end-biased histogram does not sacrifice much in the accuracy of the estimation.

6 Conclusions

Maintaining histograms to approximate frequency distributions in relations is a common technique used by database systems to limit the errors in the estimates of query optimizers. In this paper, we have examined the trade-offs between optimal histograms and histograms that can be obtained in practice. A major result of our study is that, if no information on joint-frequency matrices is available, one can decide on the histogram of each relation independently. The histogram that is optimal for the join of a relation with itself is optimal on the average for any query with any other relations with arbitrary contents. Another conclusion of our study is that end-biased histograms can be much more efficiently constructed than general serial histograms, most often without sacrificing much with respect to the estimate errors that they generate. All the above have been demonstrated in a series of experiments on both synthetic and real data.

In addition to equality join and selection queries, serial histograms turn out to be important elsewhere as well. One can easily verify that serial histograms are optimal for queries with the not-equals $\neq$ operator, since this is simply the complement of equality joins and selections. Also, range selection queries (with the $\leq, \geq, <, >$ operators) may be seen as queries with disjunctive equality selections, where the chosen selected values are those in the desired range. Thus, assuming that one is interested in both equality and range selections on a relation, serial histograms are in fact v-optimal

for queries with general selections. Based on all the above, we believe that the results presented in this paper can be used in practical systems for a wide variety of applications.

Several interesting and important questions on histogram optimality remain open. The one on top of our list deals with identifying optimal histograms for completely different types of queries (e.g., cyclic joins, non-equality joins and selections) and different parameters of interest (e.g., operator cost or ranking of alternative access plans, which determines the final decision of the optimizer). This is part of our current and future work

References

[1] C. M. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. In *Proc. of the 1994 ACM-SIGMOD Conf.*, pages 161–172, Washington, DC, May 1994.

[2] S. Christodoulakis. Estimating block transfers and join sizes. In *Proc. of the 1983 ACM-SIGMOD Conf.*, pages 40–54, San Jose, CA, May 1983.

[3] S. Christodoulakis. Implications of certain assumptions in database performance evaluation. *ACM TODS*, 9(2):163–186, June 1984.

[4] S. Christodoulakis On the estimation and use of selectivities in database performance evaluation. Research Report CS-89-24, Dept. of Computer Science, University of Waterloo, June 1989.

[5] C. Date. *A Guide to DB2.* Addison Wesley, Reading, MA, 1989.

[6] C. Faloutsos and H. V. Jagadish. On b-tree indices for skewed distributions. In *Proc. 18th Int. VLDB Conf.*, pages 363–374, Vancouver, BC, August 1992.

[7] P. Haas and A. Swami. Sequential sampling procedures for query size estimation. In *Proc. of the 1992 ACM-SIGMOD Conf.*, pages 341–350, San Diego, CA, June 1992.

[8] P. J. Haas and A. Swami. Sampling-based selectivity estimation for joins using augmented frequent value statistics. In *Proc. of the 1995 IEEE Conf. on Data Engineering*, Taipei, Taiwan, March 1995.

[9] Y. Ioannidis. Universality of serial histograms. In *Proc. 19th Int. VLDB Conf.*, pages 256–267, Dublin, Ireland, August 1993.

[10] Y. Ioannidis and S. Christodoulakis. On the propagation of errors in the size of join results. In *Proc. of the 1991 ACM-SIGMOD Conf.*, pages 268–277, Denver, CO, May 1991.

[11] Y. Ioannidis and S Christodoulakis. Optimal histograms for limiting worst-case error propagation in the estimates of query optimizers, December 1993

[12] Y. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. Unpublished manuscript, February 1995.

[13] N. Kamel and R. King. A model of data distribution based on texture analysis. In *Proc. of the 1985 ACM-SIGMOD Conf.*, pages 319–325, Austin, TX, May 1985.

[14] R. P. Kooi. *The Optimization of Queries in Relational Databases* PhD thesis, Case Western Reserve University, September 1980.

[15] R. J. Lipton, J. F. Naughton, and D. A. Schneider. Practical selectivity estimation through adaptive sampling. In *Proc. of the 1990 ACM-SIGMOD Conf.*, pages 1–11, Atlantic City, NJ, May 1990.

[16] M. V. Mannino, P. Chu, and T. Sager. Statistical profile estimation in database systems. *ACM Computing Surveys*, 20(3):192–221, September 1988.

[17] A. W. Marshall and I. Olkin. *Inequalities: Theory of Majorization and Its Applications.* Academic Press, New York, NY, 1979.

[18] T. H. Merrett and E. Otoo. Distribution models of relations. In *Proc. 5th Int. VLDB Conf.*, pages 418–425, Rio de Janeiro, Brazil, October 1979.

[19] M. Muralikrishna and D. J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *Proc. of the 1988 ACM-SIGMOD Conf.*, pages 28–36, Chicago, IL, June 1988.

[20] B. Muthuswamy and L. Kerschberg. A ddsm for relational query optimization. In *Proc. ACM Annual Conf.*, Denver, CO, October 1985.

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

[22] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the ACM-SIGMOD Conf.*, pages 23–34, Boston, MA, June 1979.

[23] Y. Wang. Experience from a real life query optimizer. In *Proc. of the 1992 ACM-SIGMOD Conf.*, page 286, San Diego, CA, June 1992. Conf. presentation.

[24] G. K. Zipf. *Human Behavior and the Principle of Least Effort.* Addison-Wesley, Reading, MA, 1949.