Probabilistic Optimization of Top N Queries

Donko Donjerkovic **Raghu Ramakrishnan**

Department of Computer Sciences
University of Wisconsin–Madison
1210 W. Dayton St.
Madison, WI 53706 USA

{donko,raghu}@cs.wisc.edu

Abstract

The problem of finding the best answers to a query quickly, rather than finding all answers, is of increasing importance as relational databases are applied in multimedia and decision-support domains. An approach to efficiently answering such "Top N" queries is to augment the query with an additional selection that prunes away the unwanted portion of the answer set. The risk is that if the selection returns fewer than the desired number of answers, the execution must be restarted (with a less selective filter). We propose a new, probabilistic approach to query optimization that quantifies this risk and seeks to minimize overall cost including the cost of possible restarts. We also present an extensive experimental study to demonstrate that probabilistic Top N query optimization can significantly reduce the average query execution time with relatively modest increases in the optimization time.

1 Introduction

In the multimedia domain, Top N or "Get the best matches" queries are common. The notion of the best match is typically fuzzy, and the cutoff (how many answers to return) is approximate, but the intent is clear. The other area where Top N queries are important is decision support, where users often want to see the high or the low end of some ordered result set. A typical example is "Find the 10 cheapest cars." The importance of Top N queries is underscored by the fact that most major commercial DBMSs include language constructs for expressing such queries. Informix supports `FIRST N`, Microsoft has `SET ROWCOUNT N`, IBM's DB2 has `FETCH FIRST N ROWS ONLY`, and Oracle supports `LIMIT TO N ROWS`.

The simplest way to support Top N queries is to execute the query, sort the result in the desired order, and then discard all but the first N tuples. Computing and sorting a large intermediate result and then discarding most of it is a waste of resources. It was shown [9] that large gains in performance are possible when the database system utilizes the fact that only a certain number of answers are needed.

A Top N query on an attribute X, denoted by Top_N^X, is equivalent to the simple selection query:

$$Top_N^X \equiv \sigma_{X>\kappa} \quad (1)$$

where κ is a *cutoff parameter* determined by N and by the data distribution. Consider the following example query on a table that is neither sorted nor indexed: "List the top 10 paid employees in the sales department". This query translates into: "List the employees from the sales department whose salary is greater than κ", where κ is determined by the distribution of employees' salaries, and must be determined by the optimizer. If κ is too high, we will retrieve less than N employees and therefore will have to restart the query with smaller κ. On the other hand, if κ is too small, the query will unnecessarily run longer. Because restarts involve repetition of work, they are characterized by a large jump in query cost.

How to estimate κ is a nontrivial problem. If the query optimizer had complete knowledge of the data distributions, it could estimate κ exactly, and eliminate restarts. However, because the optimizer's knowledge of data distributions (usually maintained in the form of histograms) is not perfect, it is better to underestimate κ as a guard against restart. The main contribution of this paper is to propose a probabilistic optimization framework that takes into account imprecision in the optimizer's knowledge of data distribution and selectivity estimates. Using probabilistic

Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

reasoning, the optimizer arrives at the *expected* cost, and the optimal cutoff parameter is the one that minimizes expected cost. While we apply the probabilistic optimization framework to the problem of estimating cutoffs for Top N queries, the approach clearly has broader applicability to optimization problems in the presence of important parameters (e.g., number of available buffers, number of concurrent queries) that can only be approximately estimated.

The rest of this paper is organized as follows. After reviewing related work, we introduce our probabilistic framework in Section 3. We introduce probabilistic optimization of Top N queries in Section 3. We develop this idea further in Section 4, where we show how to obtain selectivity and cardinality distributions for various kinds of selection predicates, starting with traditional histograms. We then present performance results for Top N queries involving selections and joins in Section 6. Next, in Section 7 we consider two classes of Top N queries that are more complex, involving aggregates and unions. The first class, involving aggregates, shows an interesting and useful connection to the class of Iceberg queries [3]. In Section 8, we then revisit the basic Top N problem formulation and identify two useful variants that can be supported using our techniques. These include an "online" variant in which answers are eagerly returned, together with some confidence bounds that they are indeed in the "top N", and a variant in which the user can specify a probability that returned answers will include all "top N", thereby controlling the time required to compute answers.

2 Related Work

Carey and Kossman [9, 1] proposed a new operator called STOP AFTER N (STOP for short) to terminate computation after the first N results are computed. Large performance gains are possible when the STOP operator is pushed down the plan tree. In contrast, while we can use the STOP operator at the root of the query sub plan, we never push the STOP operator down the plan tree. Instead, we push the equivalent selection (1), using standard techniques for handling selections. Our approach can lead to significantly better plans in some situations, as illustrated in example plans in Fig. 1. Suppose that the best plan found by the STOP pushdown is the one shown in Fig. 1 (a). Obviously, this plan can only be made cheaper by replacing the STOP operator above relation A with the equivalent selection and thus eliminating the sorting, as shown in Fig. 1 (b). Notice that the final SORT is still necessary in both versions because the hash join does not preserve sorting. *All* the implementations of STOP require at least partial sorting of the input stream, and [1] proposes techniques for reducing the sorting cost. In contrast, our approach does not require sorting, except for the final result.

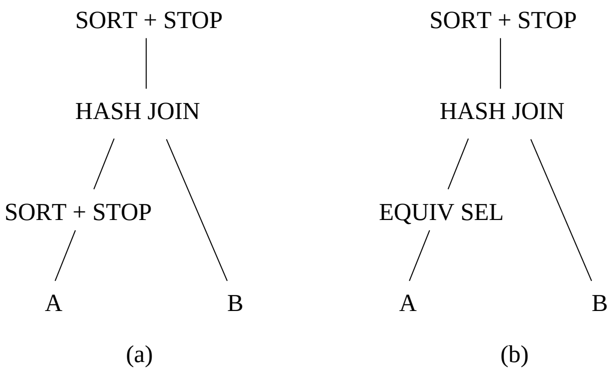

Figure 1: The plan with equivalent selection (b) may be much cheaper to execute.

Technically, the focus of our paper is on a probabilistic framework for optimization, specifically for computing the selection cutoff for Top N queries. This problem is not considered in [9, 1] or other previous work.

A concurrent work [2] has independently suggested that the optimizers should minimize expected cost of a query, and therefore have introduced the probability distributions for certain parameters. However, this work does not deal with Top N queries, which is the focus of our approach.

3 Framework for Probabilistic Query Optimization

Every Top N query is equivalent to a selection query (see Eq. (1)) with a specific cutoff parameter $\kappa = \kappa_{crit}$. Formally, κ_{crit} is defined as the largest cutoff parameter κ that does not cause restart. If complete knowledge of all selectivities and data distributions were available, restarts would never happen since one would always choose $\kappa = \kappa_{crit}$. However, since the optimizer only has approximate knowledge of distributions and selectivities, it is impossible to guarantee that more than N tuples will be eventually retrieved, short of choosing $\kappa = -\infty$. Nonetheless, we can still reason about the likelihood of restart and choose κ accordingly. To enable such probabilistic reasoning, we propose to *generalize selectivity estimates to selectivity probability distributions.*

We consider all selectivities to be random variables and denote a point in this multidimensional probability space as σ and the associated probability as $p(\sigma)$. We then postulate that the optimizers should find a plan with the minimum *expected* value of the cost $C(\sigma)$:

$$E(C) = \sum_{\sigma} C(\sigma)p(\sigma) \qquad (2)$$

Even though this work focuses on the random nature of selectivity estimates, our approach applies to other uncertain quantities that enter cost formulas such as allocated memory and connection bandwidths. In fact, the observation that optimizers should minimize expected cost was concurrently made by Chu, Halpern and Seshadri [2] and applied to the problem of memory variability.

At this point we note that, traditionally, optimizers evaluate a cost function for expected values of input parameters $C(E(\sigma))$ in the hope that this is a good approximation for overall expected value of the cost. However, it is well known that the approximation:

$$E(C(\sigma)) \approx C(E(\sigma)) \tag{3}$$

is true *only if* C is a linear function of σ within the range of variability of σ. While most of the cost formulas are not linear, in practice, they are usually well approximated by linear values in the range of variable parameters and consequently, Eq. (3) holds.

Problem of Top N optimization, is a typical example where Eq. (3) does not hold, because the jump in the total cost C due to the restart cost R is within the range of possible cutoff values. If we denote the initial cost by I, and introduce a step function ρ which is 0 when restart does not happen and 1 otherwise, we have:

$$C(\sigma) = I(\sigma) + \rho(\sigma)\, R(\sigma) \tag{4}$$

We note that the selectivity of the cutoff κ is included in the Eq. (4) as one dimension of σ. Also, Eq. (4) assumes that only one restart is possible. Even though multiple restarts could be included in this framework, it would be impractical to optimize for multiple cutoff parameters. Therefore, in our model, restart operation amounts to retrieving the complement of the equivalent selection query, sorting the result and stopping after required number of tuples is returned. Assuming that R is approximately linear in the possible selectivities of κ, we can write:

$$E(C) \approx E(I) + E(R) \sum_{\sigma} \rho(\sigma)\, p(\sigma) \tag{5}$$

$\sum_{\sigma} \rho(\sigma)\, p(\sigma)$ is the probability of restart (r) which can also be written as the probability that fewer than N answers are generated:

$$r = \sum_{n=0}^{N-1} p(n) \tag{6}$$

where $p(n)$ is the probability that the input cardinality to the Top N operator will be n.

Finally, using the traditional approximation (Eq. (3)) the objective function, parameterized by κ becomes:

$$\mathrm{Cost}(\kappa) \approx \mathrm{Init}(\kappa) + r(\kappa)\, \mathrm{Rest}(\kappa) \tag{7}$$

where Init(κ) denotes the traditional cost of processing the query with cutoff parameter κ and Rest(κ) denotes the traditional cost of processing the restart that will complete the answer to the query.

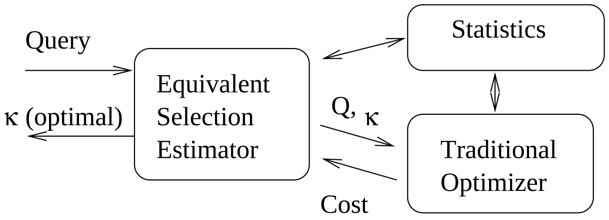

Figure 2: Architecture for incorporating κestimator into a traditional DB system.

A cutoff parameter, κ, is *optimal* if it minimizes the value of the query cost function (Eq. (7)). We restate the problem of optimizing a Top N query as the problem of finding the optimal cutoff parameter κ_{opt} and the associated execution plan. To find the minimum of the cost function (Eq. (7)) we can use a standard function minimization algorithm such as Golden Section Search [15]. The probability of restart is evaluated for every trial κ using Eq. (6).

By using the traditional approximations for expected cost values (Eq. (7)), we are able to reuse the traditional query optimizer for Top N query subtree optimization. The relationship between the equivalent selection (κ) estimator, system statistics and an optimizer are shown in Fig. 2. Equivalent selection (κ) estimator uses Golden Search to find optimal κ, and in the process calls the optimizer repeatedly to evaluate Init(κ) and Rest(κ) and consults the system statistics. For Golden search algorithm, one needs to bound the κ. Initial bound would be the column minimum for the low and the column maximum for the high. Golden search algorithm then successively splits the bound until it becomes sufficiently small.

Init(κ) and Rest(κ) are expensive expressions to evaluate because they require optimization of the query subtree. On the other hand, the best plan for Init(κ) and Rest(κ) are likely not to change for small changes in κ. Consequently, a further approximation would be to find the best plan for these two queries only once. Of course, Init(κ) and Rest(κ) should still be re-evaluated for every trial κ because the cost will change depending on κ even if the plan does not change.

3.1 Probability Distribution Maintenance

In this section we describe how to practically maintain cardinality distributions; the ideas apply to maintaining selectivity distributions as well. In general, a cardinality distribution is completely specified by $(cardinality - value, probability)$ pairs, but maintaining all such pairs is not practical. A simple approximation is to only store a certain number of cardinality values whose associated probabilities are all the same.

For example, a selectivity vector of size η could be represented as an array:

$$\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_\eta\}$$

where σ_i are all equally probable selectivities. By choosing this alternative we don't have to store individual probabilities, since they are all the same and equal to $1/\eta$. The size of the probability vector (η) is system dependent. A selectivity distribution can be represented in a similar manner.

To find the result of multiplying a cardinality distribution with a selectivity distribution, we just multiply every possible selectivity with every possible cardinality. However, the resulting distribution will have η^2 elements and must be reduced to only η elements; this approximation can be carried out by replacing η neighboring values with their average.

4 Estimating Initial Probability Densities

We have discussed how to propagate cardinality densities through the plan tree, by multiplying the operator selectivity and the input cardinality densities. However, we have not yet addressed the problem of estimating the *initial* cardinality density and the *initial* selectivity density for every predicate in the query; we turn to this next. Database systems usually maintain exact cardinalities for the base tables. Therefore, initial cardinality densities are likely to be single values with probability one. Estimating selectivity densities is much more complex. Keeping in mind that our estimates will be used for optimization purposes only, precision is not of crucial importance, so we choose simplicity as our guiding principle.

We will estimate initial selectivity distributions from histograms. In order for the selectivity distribution to be *consistent* with the traditional (single value) histogram estimate, we require that the expected value of the selectivity distribution coincide with the traditional selectivity estimate. [1] Therefore, we propose to construct a selectivity distribution whose average is equal to the traditional selectivity for a predicate, call it σ. As described in Sec. 3.1, our distribution consists of a set of equally probable cardinality values. Finally, we need to bound our distribution to the left and to the right. Distribution spread reflects the precision of the histogram estimates; the more accurate the histogram is the tighter the bounds.

Summarizing these ideas, we arrive at the generic distribution shown in Fig. 3. Notice that, in general, the left bound (B_L) need not be equal to the right bound (B_R). For example, bounds for a predicate can be asymmetric because a predicate selectivity may not exceed one nor be less than zero. Given the average value (traditional estimate σ from a histogram) and bounds (B_L and B_R) one can easily construct a simple distribution with a certain number of possible values located equi-distantly to the left of the average and the remaining values positioned equi-distantly to the right. Equi-distant positioning is chosen for simplicity, notice that the distance between the left-hand side values may not be the same as the corresponding distance on the right. The total number of values in a selectivity distribution is a predetermined constant (we used 32 in our experiment). Number of values to the left of σ is calculated so that the expected value of all the distribution is equal to σ. In the following sections we will discuss how to estimate the two distribution parameters B_L and B_R for common predicates.

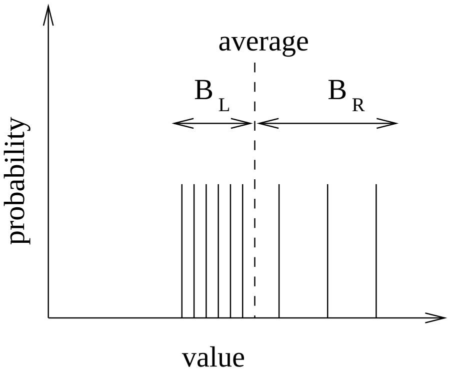

Figure 3: Example of an initial selectivity density.

4.1 Estimating the Quality of a Histogram

Distribution parameters B_L and B_R are dependent on the quality of the histogram on the referenced column. Research on histograms has mainly focused on improving their precision [10]. The first paper to introduce the idea of augmenting a histogram with some measure of accuracy is [5]. They suggest maintaining the largest equality selection error within each bucket. This error is determined by comparing histogram estimates to the actual result of an equality selection.

Although the idea of maintaining some error estimates within a histogram is a good one, maintaining per bucket information has the following disadvantages: (1) Per bucket error information will increase the size of the bucket and therefore use space that could otherwise be used to increase histogram precision. (2) Selection errors for range queries will be largely overestimated if they are based on the largest errors per bucket. This is because errors in single values tend to cancel each other, and simply adding them up will greatly overestimate the error.

We propose to maintain the worst-case error for an open-ended range predicate. This has an advantage

[1] Given a predicate, say $X < 100$, its selectivity is estimated from a histogram on the data distribution by adding counts in buckets to the left of the point $X = 100$ and taking the ratio to the total count over all buckets.

of requiring little space, independent of the number of buckets, and it provides good bounds for queries of type $field \leq value$. More specifically, let x denote the domain values, $P_{\text{real}}(x)$ denote the cumulative probability distribution of the real data set and $P_{\text{hist}}(x)$ denote the cumulative probability distribution deduced from a histogram. Then, we define ϵ as:

$$\epsilon = \max_{-\infty < x < \infty} \mid P_{\text{real}}(x) - P_{\text{hist}}(x) \mid \qquad (8)$$

In other words, ϵ is the maximum deviation of the selectivity of the predicate $field \leq value$ between the histogram and the real data set. We propose to experimentally measure ϵ for each histogram and maintain this value as a part of the system statistics. Notice that a table without a histogram is usually assumed to have uniform distribution that corresponds to the trivial histogram, with only one bucket. Therefore, without the loss of generality, we consider every table to have an associated histogram.

The most precise (and the most expensive) way of measuring ϵ is by sorting the original table and performing the full scan. A much cheaper way is to take a random sample of the original table and measure ϵ from the random sample. The crucial question here is how big a sample is needed in order to estimate ϵ correctly. In general, this depends on the precision of the histogram: the more precise the histogram is, the larger the required sample. Histogram precision in turn depends on the type of the histogram and on the number of buckets β. The most commonly used histogram in current database systems is the equi-depth histogram, and so we present a short analysis for it here. The value of ϵ for an equi-depth histogram is bounded as:

$$\epsilon \leq \frac{1}{\beta} \qquad (9)$$

where β is the number of buckets. Also, by the theorem due to Kolmogorov [7] we have:

$$D \leq \frac{\lambda}{\sqrt{s}} \qquad (10)$$

where s is the size of the random sample, D is the maximal deviation between the real data set and its sample (Eq. (8)), and λ is a number that depends on the confidence limit. For 80% confidence, $\lambda \approx 1$. So, the pessimistic estimate of D for 80% confidence is:

$$D \approx \frac{1}{\sqrt{s}} \qquad (11)$$

To reliably estimate ϵ, D should be much smaller than ϵ, say

$$D \approx \frac{\epsilon}{10}. \qquad (12)$$

From formulas (9), (11), and (12) it follows that s can be approximated by:

$$s \geq 100\,\beta^2 \qquad (13)$$

We have verified experimentally that the sample size of approximately $100\,\beta^2$ produces satisfactory results. (See Fig. 10).

Notice that ϵ can be calculated at the histogram construction time, using the single sample for both, building the histogram and estimating ϵ. In fact, the required sample size is, for the most cases, of the same order of magnitude. For example, a histogram with 100 buckets ($\beta = 100$) would require a sample of size of 1 million (Eq. (13)). On the other hand, a recent paper on equi-depth histogram construction [13] suggests that for the reasonable values of confidence, data size and deviations from true equi-depth histogram, 0.8 million is the recommended sample size.

4.2 Estimating Selectivity Probability Density for Open Range Selection

From the definition of ϵ (Eq. (8)) and the definition of the cumulative probability density it is clear that the maximal error in the open range selection is ϵ. Therefore, we construct a selectivity density shown in Fig. 3 with the average equal to the selectivity estimate from the histogram and $B_L = B_R = \epsilon$.

4.3 Estimating Selectivity Probability Density for Equality and Closed Range Selection

By knowing ϵ, one can bound the error in an equality selection as well. If one denotes the histogram error in the frequency of a domain value i by Δf_i then the following condition must hold:

$$-\epsilon \leq \sum_{i=-\infty}^{j} \Delta f_i \leq \epsilon \qquad (14)$$

for any j element of the value domain. One can express the error in frequency Δf_j as:

$$\Delta f_j = \sum_{i=-\infty}^{j} \Delta f_i - \sum_{i=-\infty}^{j-1} \Delta f_i$$

from which it is seen than Δf_j is bounded as:

$$-2\epsilon \leq \Delta f_j \leq 2\epsilon \qquad (15)$$

Following the same argument, it can be shown that the error in the cardinality result R of the closed range query (like $a \leq x \leq b$) is bounded by:

$$-2\epsilon \leq \Delta R \leq 2\epsilon \qquad (16)$$

i.e., it is independent of the range. Similar to the open range selection, we construct a selectivity density shown in Fig. 3 with the average equal to the selectivity estimate from the histogram and $B_L = B_R = 2\epsilon$.

4.4 Estimating Selectivity Probability Density for Equi-join Selection

The resulting cardinality of an equi-join (R) can be expressed as:

$$R = \sum_i f_i g_i \tag{17}$$

where f and g stands for the frequency vectors of the two tables to be joined and i ranges over all domain values in the join columns. Error in R can be obtained by differentiating Eq. (17):

$$\Delta R = \sum_i \Delta f_i \, g_i + \sum_i f_i \Delta g_i \tag{18}$$

where we have ignored the term $\sum_i \Delta f_i \, \Delta f_j$ because it is small compared to the other terms. This expression can be further simplified by rewriting:

$$\begin{aligned} f_i &= \tilde{f}_i + \Delta f_i && (19) \\ g_i &= \tilde{g}_i + \Delta g_i && (20) \end{aligned}$$

where $\tilde{f}_i$ and $\tilde{g}_i$ stand for the histogram estimate of f_i and g_i respectively. After substituting the above expressions into Eq. (18) and ignoring the terms with two differentials we get:

$$\Delta R \approx \sum_i \Delta f_i \, \tilde{g}_i + \sum_i \tilde{f}_i \Delta g_i \tag{21}$$

or by noticing that $\tilde{f}$ (and $\tilde{g}$) is constant within a bucket b:

$$\Delta R \approx \sum_b \tilde{g}_b \sum_{j \in b} \Delta f_j + \sum_b \tilde{f}_b \sum_{j \in b} \Delta g_i \tag{22}$$

Finally, using the bounds from Eq. (16) we obtain:

$$\Delta R \leq 2\epsilon_f \sum_b \tilde{g}_b + 2\epsilon_g \sum_b \tilde{f}_b \tag{23}$$

From these bounds, we construct a selectivity density shown in Fig. 3 with the average equal to the selectivity estimate from the histogram and $B_L = B_R = \Delta R$.

4.5 Estimating Selectivity Probability Density for Selections on Union

We examine the issues related to Top N queries over unions motivated by the following observations:

1. Many database integration systems, which are expected to have significant presence on the Web, are built as unions over the base tables (see for example [8] and [11]).

2. Top N queries are one of the most common queries in the Web environment. We will then especially be concerned with running a Top N query on a distributed union.

Maximum error in the resulting cardinality ΔR of a selection on union is just the sum of all the component errors ΔR_i.

$$\Delta R = |\Delta R_1| + |\Delta R_2| + \ldots + |\Delta R_n| \tag{24}$$

From this bounds, we construct a selectivity density shown in Fig. 3 with the average equal to the selectivity estimate from the histogram and $B_L = B_R = \Delta R$.

5 Example

Assume that we want the salaries of top 50 paid employees whose age is less than 40. Selectivities presented in the following table were determined from the system statistics using standard estimation techniques.

Predicate	Selectivity	Max Error
$age < 40$	0.4	0.2
$salary > 100K$	0.1	0.3

Maximal errors for the open range selections was measured and stored with other system statistics. Suppose that the Golden Search technique is currently trying to evaluate the cost function (Eq. (7)) for cutoff $\kappa = 100K$. Assuming that the system is configured with $\eta = 4$, we construct the initial distribution for age predicate, as shown in Fig. 4. Similarly, initial distribution for $salary > 100K$ is shown in Fig. 5. In general, the number of columns to the left of the average η_L and to the right of the average η_R is determined by the following equations:

$$\begin{aligned} \eta_L B_L &= \eta_R B_R \\ \eta_L + \eta_R &= \eta \end{aligned}$$

Result of multiplying these two selectivity distribution, multiplied by the total number of input tuples (1,000), is shown in Fig. 6. From Fig. 6 we conclude that the probability of restart for $\kappa = 100K$ is 75% because 3 out of 4 columns are less than 50. In a similar manner, one would continue with the next iteration of κ and stop when the minimum is bounded with sufficient precision (e.g., 1/10 of the bucket width).

6 Performance Evaluation for Selection and Join Queries

In the following sections, we have applied the ideas developed so far to the optimization of Top N queries on a single table or a join. We compare execution times for the following three algorithms, using average execution time for 15 randomly generated input data sets:

Traditional: Compute all answers, sort, and return the top N.

Naive: Estimate the cutoff parameter for top $1.2\,N$ (20% safety margin) using available system statistics.

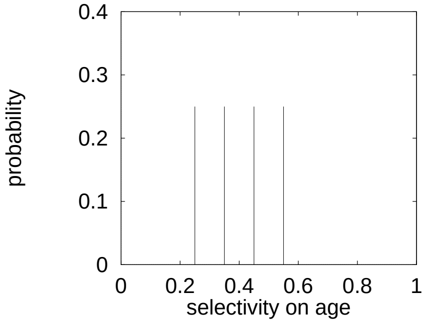

Figure 4: Selectivity distribution of predicate $age < 40$.

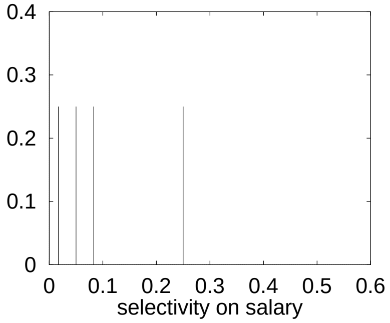

Figure 5: Selectivity distribution of predicate $salary > 100K$.

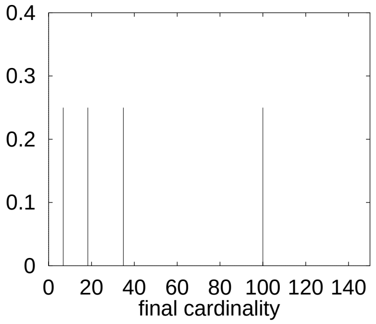

Figure 6: Output cardinality distribution.

Probabilistic: Determine the cutoff parameter probabilistically, using available system statistics (including the measured ϵ).

We varied several parameters: (1) Skew of the underlying data distribution (Zipf parameter[16] Z, by default one). (2) Number of buckets in the histogram. (3) N, the number of tuples selected, by default 1,000. (4) s, the size of the random sample used to estimate ϵ. We fixed the total number of tuples in the data file (100,000), and the total spread of the data, which is approximately equal to the number of distinct values (5,000). We estimated execution times by using standard analytical formulas for cost estimation [12], estimating the cost of a disk I/O as $10ms$ and the CPU cost of a tuple swap (in sorting) as $10\mu s$. Our results show the performance gains to be sufficiently large that the relative merits of our probabilistic approach hold regardless of the approximations inherent in this simple estimation of execution time.

6.1 Top N on a Single Table Selection Query

Consider the query that asks for the Top N employees by salary. Assume that the *Employees* table is neither sorted nor indexed on *salary* field. As suggested by [1], the best plan for this query is probably to use range-partitioning sort. However, the crucial question is how many partitions to materialize. In order to simplify our presentation, we consider only two partitions, one which is materialized and sorted and the other with the rest of the data. (In the terminology of the paper [1] these two partitions are called the winner and the loser, respectively.) In the case of multiple (memory-sized) partitions, there will still be two large groups, one that contains materialized partitions and the other that contains unmaterialized ones. Therefore, our simplified analysis and conclusions would still hold in the more complex multi-partition case. We discuss the parameters varied and the corresponding figures next.

Data Skew: Fig. 7 has the number of histogram buckets fixed to one, implying the uniformity assumption. When data is really uniform ($Z = 0$), the naive and the probabilistic algorithm have the same performance. With a large data skew, uniformity assumption becomes significantly violated and the naive algorithm frequently runs into restarts. Notice that restarts are more expensive that the traditional scan + sort approach. The probabilistic algorithm handles skew gracefully by just becoming more pessimistic in choosing the cutoff.

Number of Buckets: Fig. 8 shows that as the number of buckets increases, the difference between the probabilistic and the naive algorithm becomes less pronounced. This is due to the fact that with a larger number of buckets, the histogram error falls below 20% in which case the naive algorithm will not restart.

Top N selected: Fig. 9 shows that the naive and the probabilistic algorithm converge as N increases. This is because of the fact that eventually the 20% overestimate becomes adequate (conservative), provided that N is large enough. For small N, 20% obviously does not provide enough safety margin.

Sample Size: Fig. 10 shows that the sample size of 100 or more (as predicted by Eq. (13)) is satisfactory for this experiment, and that the performance of the probabilistic algorithm is not sensitive to small variations in the sample size.

6.2 Top N on Equi-Join Queries

Consider an equi-join query of two identical tables that have on average 20 duplicates for each value in the join column, augmented by Top N operator on an independently distributed column. In this section, we compare the performance of naive and probabilistic algorithms on equi-join queries such as this. We used the same data generator as for the selection queries, which implies that the average number of duplicates for a certain attribute value is 20. We discuss the parameters varied and the corresponding figures next.

Data Skew: Fig. 11 shows the increased gap in performance as the data skew increases initially, due to the fact that the naive algorithm runs into restarts. Restarts for the Naive algorithm become more common for increasing skew because the histogram esti-

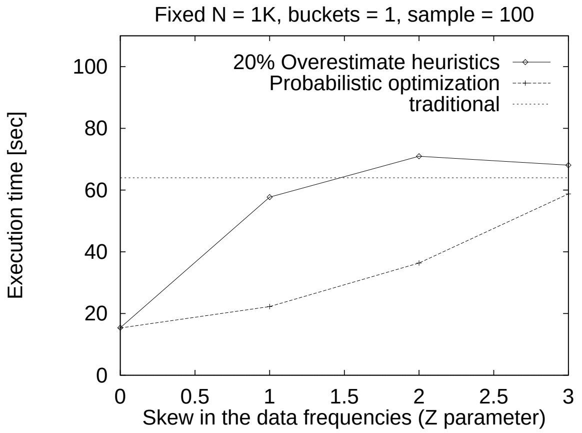

Figure 7: Execution time vs. data skew, using trivial (1 bucket) histogram.

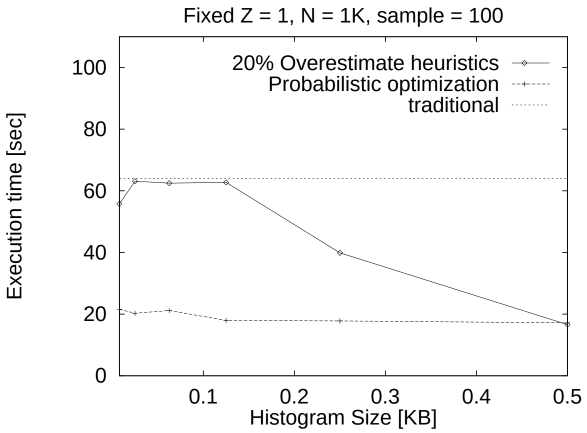

Figure 8: Execution time vs. number of buckets in histogram.

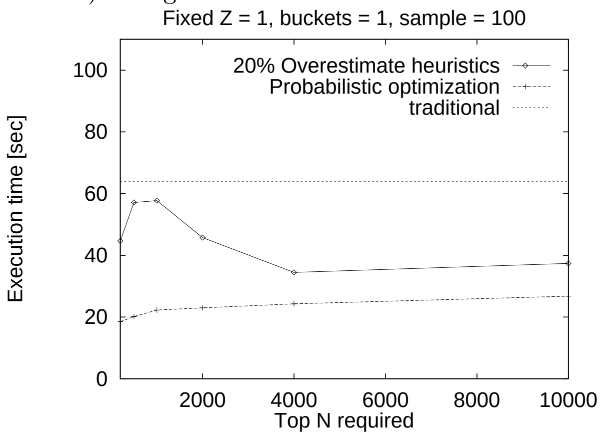

Figure 9: Execution time for different values of Top N selected (in percents of relation size).

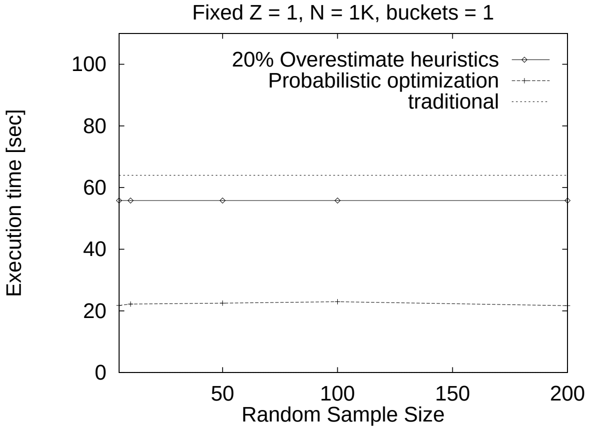

Figure 10: Execution time vs. sample size used to calculate ϵ.

mates become increasingly unreliable. However, algorithms converge for the extreme skews because the result of the equi-join query goes to zero (no matches) and both algorithms select the whole result (N is larger than the result size).

Size of Histogram: Fig. 12 shows that the naive algorithm improves as the histograms become larger, as expected. The probabilistic algorithm improves too but the trend is too small to be visible.

Top N Selected: Fig. 13 shows that the differences between algorithms are less pronounced when larger N is selected, because the 20% overestimate becomes adequate for larger N. The reasoning here is the same as in single table case.

Number of Joins: Fig. 14 shows that the naive algorithm does not work for more than 2 way joins on the test data. The reason for this is twofold. First, the quality of the estimates deteriorates rapidly with the number of joins, thus making the restarts more likely. Second, the punishment for restart skyrockets due to the large join size (100,000 * 20 * 20 tuples for the 3-way join).

In general, join experiments reflect the fact that estimating join selectivity is much more difficult than estimating selectivity of range predicates [4], and consequently, the probabilistic approach is of greater value in this case.

7 Improvements on Some Common Top N Query Evaluations

In this section we consider two cases in which significant additional improvements over the standard Top N query processing are possible: Top N on aggregate queries and Top N over distributed unions.

7.1 Efficient Evaluation of Top N Queries on Aggregates

Consider a Top N aggregate query such as this one asking for the N most common ages among employees:

select age, count(age) **from** Employees emp
group by age **order by** count(age)
stop after N

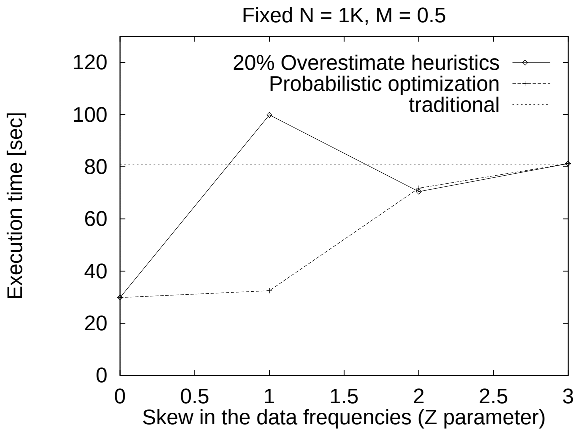

Figure 11: Execution time vs. data skew.

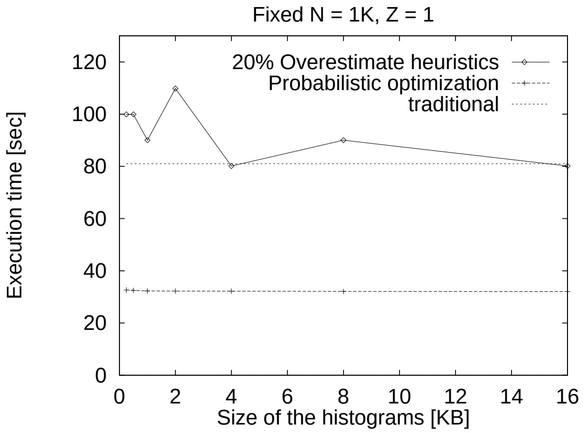

Figure 12: Execution time vs. number of buckets in the histogram.

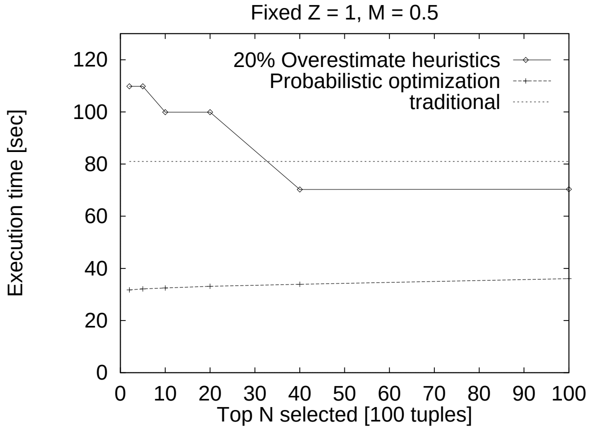

Figure 13: Execution time for different values of Top N selected (in thousand of tuples).

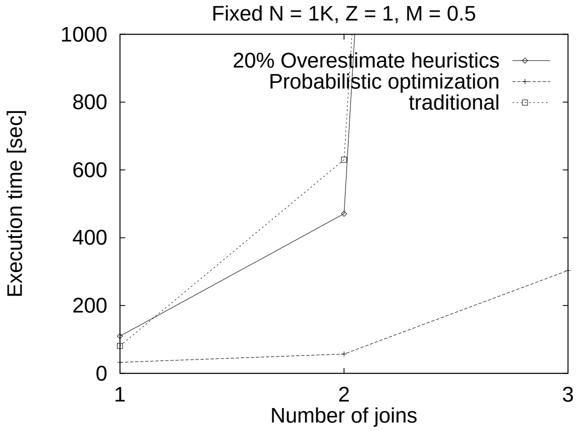

Figure 14: Execution time dependency on the number of joins.

Given a small candidate set of "frequent" ages, we can scan the data to compute accurate frequency counts, maintaining one main memory counter per candidate age, and then select the top N by frequency. The main problem is to identify a small set of frequent age values that includes the top N ages by frequency. We discuss two alternative evaluation strategies.

(I) Reduction to an Iceberg Query: The idea is to replace the Top N operator by the equivalent selection. We need to estimate the cutoff value κ for count(age), then group employees by age and compute the counts above the cutoff. Given the cutoff κ, we can turn the above Top N query into an Iceberg query, allowing us to use the algorithms proposed in [3], as follows: just replace the **stop after** clause with **having** count(age) $> \kappa$. Using this approach, the algorithms of [3] require two full scans of the dataset (one to identify the "frequent" ages, and one to compute their counts), and there is the possibility of additional scans in the case of restart (due to the Top N nature of our main query).

(II) Direct Use of a Histogram: This approach requires a histogram on the Top N attribute (*age* in this example). Let the largest error in equality selection on this histogram be E. Using the histogram, choose an attribute value V that has the smallest frequency F among the N attribute values with the largest frequencies. The actual dataset may have a frequency for value V that is as low as $F-E$. Also, other frequencies in the histogram may be underestimated, and so the candidate set (for inclusion in the Top N) is any value whose histogram frequency is above $F-2E$. The existence of a histogram therefore allows us to identify a candidate set of frequent attribute values that is *conservative*: the top N values by frequency are guaranteed to be here (provided that the error bounds stored with the histogram are accurate!). This eliminates the problem of restart, and further, the candidate set generation is based purely on the histogram. The database is scanned once to count frequencies for each candidate "frequent" attribute value. In Fig. 15 we present experimentally measured number of candidates for the example Top N query on a synthetically

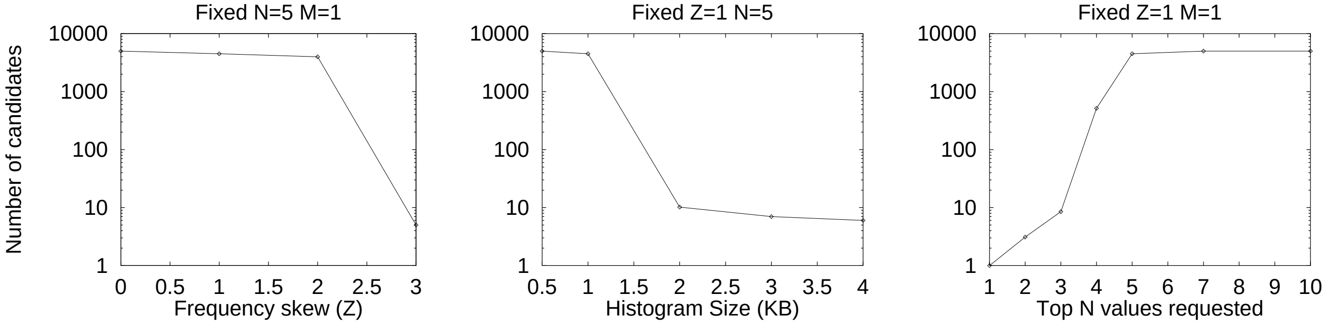

Figure 15: Number of candidates generated by the direct histogram usage as a function of data skew, histogram size, and number of tuples requested.

generated data set. The three graphs in Fig. 15 show expected trends in the effectiveness of the direct histogram alternative, which can be summarized as follows:

1. Number of candidates decreases as the data skew increases. This is expected behavior since it is easier to identify the Top N candidates when there are large differences among frequencies.
2. Number of candidates decreases as the histogram precision (size) increases. This is because the error decreases when the size is increased, making the candidate threshold frequency $F - 2E$ higher.
3. Number of candidates exponentially increases with N (number of tuples requested). This is mainly an artifact of the Zipf distribution, which is exponential.

The conclusion of this section is that the direct histogram method of finding the candidate set is an excellent way to answering Top N queries on aggregates under the circumstances of high skew, large histograms ($> 1KB$), and small N.

7.2 Lazy Evaluation of Top N Over Distributed Unions

In a distributed environment, a Top N query could be run in parallel, ensuring the shortest response time. However, this may unnecessarily waste the computing resources of remote sites. We can reduce resource consumption by waiting to access a new site until it is necessary to do so, at the cost of slowing the execution.

If the user chooses to conserve the resources, what is the proper order of accessing the sites so that the number of accessed sites is minimal? We propose to access the sites in the order of estimated probabilities that they will be useful in answering the query. Suppose that at a certain site S the maximum value for the field of interest is M_S. If M_S is less than the cutoff parameter κ, we will certainly not access the site S. However, even if $\kappa \leq M_S$ there is still a chance that the site S will not be accessed because the κ might be underestimated. The probability of accessing the site S is the probability of restart when $\kappa = M_S$. (The Top N query is translated to selection above the cutoff parameter.) In other words, if $\kappa = M_S$ and no restart occurs than the site S need not be accessed. So, the sites should be accessed in the order of the decreasing probability of being needed. Because the probability of restart is a monotonically decreasing function of the cutoff parameter, this order coincides with the order of decreasing M_S. The benefits of the lazy approach can be potentially large, as shown in Fig. 16. The reduction of the resource usage for certain values of N is due to the fact that one connection to the remote source was saved. In this experiment, we used a union with 20 members whose data are identically but independently distributed.

8 Useful Variants of Top N Queries

8.1 Online Top N with Confidence Estimates

Motivated by the ideas of Online Aggregation [6], we consider an online version of the Top N operator. Online operators are characterized by providing (1) approximate answers that are periodically updated, and (2) some probabilistic guarantees about the (degree of) correctness of the current answers. An online Top N operator should therefore provide a set of N or fewer answers that are likely to be in the Top N list, along with associated probabilities indicating the likelihood that a given answer will be in the final Top N list.

Our probabilistic framework provides the infrastructure to implement such an operator. Consider, for example, a Top N query on a single table. The system will periodically display the current set of tuples that satisfy the cutoff predicate. The probability of a value x not being in the Top N results is the probability of no restart happening when $\kappa = x$. Equivalently, the probability of a selected value x being in the final Top N values is the probability of restart when $\kappa = x$, where the probability of restart is calculated using Eq. (6). These probabilities do not depend on the order in which the data is read.

In the event of restart, while getting all N results

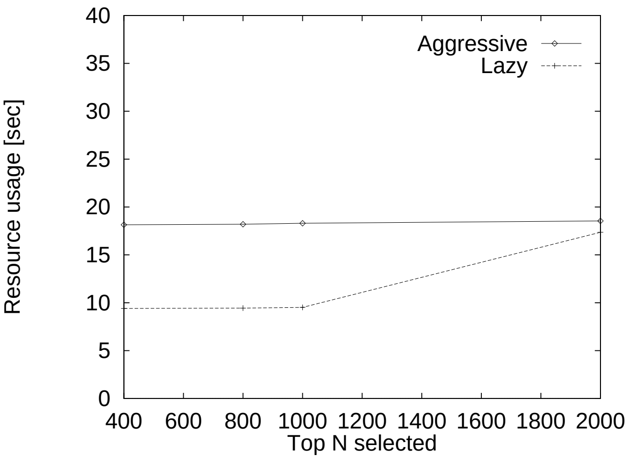

Figure 16: Total resource usage for a union consisting of 20 members with trivial histograms.

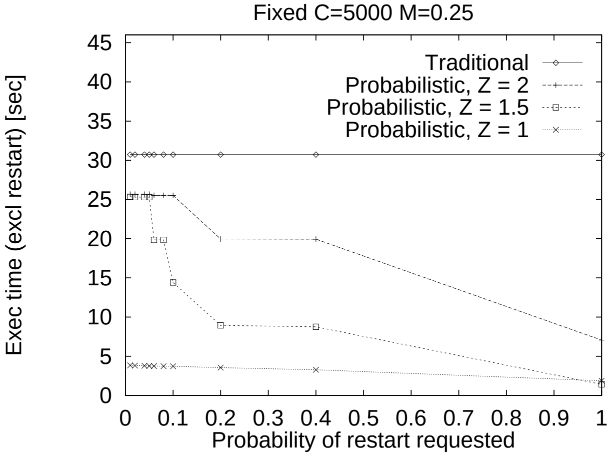

Figure 17: Execution time dependency on user-specified restart probability for single table scans

will take longer, the user at least has a subset of K results which, as of the time restart is initiated, are guaranteed to be the top K. If K is sufficiently close to N, the user may well terminate computation at this point (after all, the choice of N is likely to be rather ad hoc in the first place).

8.2 Fuzzy Top N: An Alternative Formulation of Top N

Top N queries require exactly N answers, and the system has to guarantee N results by restarting the query if necessary. We observe that many times, users may not insist on exactly N answers but may be ready to accept less. We formalize this intuition by allowing a user to specify a bound on the likelihood of restarts. So if a user is willing to accept a small likelihood of restart, the system can compute the cutoff κ more aggressively, and find answers in less time. Of course, as κ is set more and more aggressively, the likelihood of restart increases, and intuitively, the number of answers computed as of the time of restart decreases. So the user indirectly also controls the number of answers that are likely to be computed at the time of restart by directly controlling the bound on the likelihood of restart.

In this formulation of the problem, the cutoff κ is determined solely by p and N (and of course data distribution) but not by the estimated execution time. The desired cutoff is such that it minimizes $|r - p|$ where r is the probability of restart (defined in Eq. (6)) and p is the probability of calculating N or more answers (given by the user). For minimization one can again use the Golden Search technique. After this cutoff is determined, we could just use a traditional optimizer to optimize the query augmented with equivalent selection. This makes it very easy to support Fuzzy Top N in an existing system; all that is needed is a thin layer (using the probabilistic estimation techniques presented here) to augment a query with a cutoff selection predicate.

We have experimentally measured the query execution times (not including restart) for various restart probabilities requested and the skew of the input data. In Fig. 17 we show the results for the single table Top N query for input data files of 100,000 tuples spread over attribute range of 5,000 distinct values. The top 10,000 answers were requested and the histogram size was fixed to 0.25 KB. For comparison, we also include the time for the Traditional alternative which would sort all the data and return first N tuples only. Fig. 17 indicates that for low skews the execution time is not very dependent on the probability of restart. This is due to the fact that a 0.25KB compressed histogram can bound the possible cutoff values well within a small range of attribute values. On the other hand, datasets with high skew require much longer execution time for low values of the probability of restart. This can be explained by the fact that with high skew there are certain attribute values that make up the bulk of the distribution. Selecting such a value ensures no restart with certainty and not selecting it ensures a restart with certainty. When choosing between zero and one, the system chooses zero for small restart probabilities, effectively selecting and sorting large chunks of input data.

9 Future Work

We plan to examine the benefits of the probabilistic optimization for traditional select-project-join queries. Probabilistic query optimization should reduce the average execution time in cases when plan's cost is not a linear function of resources that vary within the non-linear region. Example of such cases are the join cost formula non-linear dependency on the available memory. Another example is the problem of executing queries that refer to relations scattered over a widearea network [14]. The challenge here is to come up with plans whose execution times are not too sensitive to the possible delays in the network. Yet another example can be found in distributed query processing, where the optimizer has to distribute the jobs to the sites depending on the machine loads.

10 Conclusion

We have presented a new solution to the optimization of Top N queries that offers an interesting, and in some ways simpler, alternative to the approach of [9, 1]. Our extensions to a traditional query optimizer are relatively easy to implement and they show significant improvements in execution times over the naive approach to aggressive pushing of STOP operator. The underlying idea of taking imprecision in estimates into account during query optimization has much wider applicability than just Top N queries.

References

[1] Michael J. Carey and Donald Kossmann. Reducing the braking distance of an sql query engine. In *Proceedings of the International Conference on Very Large Data Bases*, 1998.

[2] P. Seshadri F. Chu, J. Halpern. Least expected cost query optimization: An exercise in utility. In *Proceedings of the International Conference on Very Large Data Bases*, 1999.

[3] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman. Computing iceberg queries efficiently. In *Proceedings of the International Conference on Very Large Data Bases*, pages 299–310, 1998.

[4] Yannis E. Ioannidis and Stavros Christodoulakis. On the propagation of errors in the size of join results. In *Proceedings of ACM-SIGMOD Conference on Management of Data*, 1991.

[5] H.V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Ken Sevick, and Torsten Suel. Optimal histograms with quality guarantees. In *Proceedings of the International Conference on Very Large Data Bases*, 1998.

[6] Helen J. Wang Joseph M. Hellerstein, Peter J. Haas. Online aggregation. In *Proceedings of ACM-SIGMOD Conference on Management of Data*, Tucson, Arizona, 1997.

[7] A. N. Kolmogorov. Confidence limits for an unknown distribution function. In *Ann. Math. Statist.*, pages 461–463, 1941.

[8] Alon Levy, Anand Rajaraman, and Joann Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the International Conference on Very Large Data Bases*, 1996.

[9] Donald Kossmann Michael J. Carey. On Saying “Enough Already!” in SQL. In *Proceedings of ACM-SIGMOD Conference on Management of Data*, Tucson, Arizona, 1997.

[10] Viswanath Poosala, Yannis Ioannidis, Peter Haas, and Eugene Shekita. Improved histograms for selectivity estimation of range predicates. In *Proceedings of ACM-SIGMOD Conference on Management of Data*, pages 294–305, June 1996.

[11] Raghu Ramakrishnan and Avi Silberschatz. Scalable integration of data collection on the web. In *Technical Report: CS-TR-98-1376.* University of Wisconsin-Madison, June 1998.

[12] Leonard D. Shapiro. Join processing in database systems with large main memories. In *ACM Transactions on Database Systems*, volume 11, pages 239–264, 1986.

[13] Vivek R. Narasayya Surajit Chaudhuri, Rajeev Motwani. Random sampling for histogram construction: How much is enough? In *Proceedings of ACM-SIGMOD Conference on Management of Data*, pages 436–447, 1998.

[14] Tolga Urhan, Michael J. Franklin, and Laurent Amsaleg. Cost based query scrambling for initial delays. In *Proceedings of ACM-SIGMOD Conference on Management of Data*, pages 130–141, 1998.

[15] Saul A. William H. Press, Brian P. Flannery and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1993.

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