ISOMER: Consistent Histogram Construction Using Query Feedback

U. Srivastava[1] P. J. Haas[2] V. Markl[2] N. Megiddo[2] M. Kutsch[3] T. M. Tran[4]

[1] Stanford University usriv@stanford.edu

[2] IBM Almaden Research Center {phaas,marklv,megiddo}@almaden.ibm.com

[3] IBM Germany kutschm@de.ibm.com

[4] IBM Silicon Valley Lab minhtran@us.ibm.com

Abstract

Database columns are often correlated, so that cardinality estimates computed by assuming independence often lead to a poor choice of query plan by the optimizer. Multidimensional histograms can help solve this problem, but the traditional approach of building such histograms using a data scan often scales poorly and does not always yield the best histogram for a given workload. An attractive alternative is to gather feedback from the query execution engine about the observed cardinality of predicates and use this feedback as the basis for a histogram. In this paper we describe ISOMER, a new feedback-based algorithm for collecting optimizer statistics by constructing and maintaining multidimensional histograms. ISOMER uses the maximum-entropy principle to approximate the true data distribution by a histogram distribution that is as "simple" as possible while being consistent with the observed predicate cardinalities. ISOMER adapts readily to changes in the underlying data, automatically detecting and eliminating inconsistent feedback information in an efficient manner. The algorithm controls the size of the histogram by retaining only the most "important" feedback. Our experiments indicate that, unlike previous methods for feedback-driven histogram maintenance, ISOMER imposes little overhead, is extremely scalable, and yields highly accurate cardinality estimates while using only a modest amount of storage.

1. Introduction

Query optimization relies heavily on accurate cardinality estimates for predicates involving multiple attributes. In the presence of correlated attributes, cardinality estimates based on independence assumptions can be grossly inaccurate [15], leading to a poor choice of query plan. Multidimensional histograms [14, 15] have been proposed for accurate cardinality estimation of predicates involving correlated attributes. However, the traditional method of building these histograms through a data scan (henceforth called the *proactive* method) does not scale well to large tables. Such a histogram needs to be periodically rebuilt in order to incorporate database updates, thus exacerbating the overhead of this method. Moreover, since proactive methods inspect only data and not queries, they generally do not yield the best histogram for a given query workload [4].

In this paper, we consider an alternative method [1, 4, 6, 11] of building histograms through query feedback (henceforth called the *reactive* method). For example, consider a query having a predicate `make = 'Honda'`, and suppose that the execution engine finds at runtime that 80 tuples from the `Car` table satisfy this predicate. Such a piece of information about the observed cardinality of a predicate is called a query-feedback record (QFR). As the DBMS executes queries, QFRs can be collected with relatively little overhead [17] and used to build and progressively refine a histogram over time. For example, the above QFR may be used to refine a histogram on `make` by creating a bucket for `'Honda'`, and setting its count to 80.

The reactive method is attractive since it scales extremely well with the table size, does not contend for database access, and requires no periodic rebuilding of the histogram (updates are automatically incorporated by new QFRs). The histogram buckets may be chosen to best suit the current workload, thereby efficiently focusing resources on precisely those queries that matter most to the user. Of course, the reactive method may lead to inaccuracies for parts of the data that have never been queried. However, such errors can be ameliorated, for example, by starting with a coarse histogram built proactively [4].

Previous proposals for histogram construction using query feedback have lacked either accuracy or efficiency. Some proposals, e.g., STGrid [1], use heuristics to refine the histogram based on a new QFR, thereby leading to inaccuracies in the constructed histogram. Other proposals, e.g., STHoles [4], require extremely detailed feedback from the query execution engine that can be very expensive to gather at runtime. In this paper, we describe ISOMER (Improved Statistics and Optimization by Maximum-Entropy Refinement), a new algorithm for feedback-driven histogram construction. In contrast to previous approaches, ISOMER is both accurate as well as efficient. ISOMER uses the

information-theoretic principle of *maximum entropy* [10] to approximate the true data distribution by a histogram distribution that is as "simple" as possible while being consistent with the observed cardinalities as specified in the QFRs. In this manner, ISOMER avoids incorporating extraneous—and potentially erroneous—assumptions into the histogram.

The reactive approach to histogram maintenance entails a number of interesting challenges:

1. **Enforcing consistency**: To ensure histogram accuracy, the histogram distribution must always be consistent with all currently valid QFRs and not incorporate any ad hoc assumptions; see Section 3.2. Previous proposals for feedback-driven histogram construction [1, 11] have lacked this crucial consistency property, even when the data is static.

2. **Dealing with data changes**: In the presence of updates, deletes, and inserts, some of the QFRs collected in the past may no longer be valid. Such old, invalid QFRs must be efficiently identified and discarded, and their effect on the histogram must be undone.

3. **Meeting a limited space budget**: Database systems usually limit the size of a histogram to a few disk pages in order to ensure efficiency when the histogram is read at the time of query optimization. Thus, we assume that there is a limited space budget for histogram storage. In general, adding more QFRs to the histogram while maintaining consistency leads to an increase in the histogram size. To keep the histogram size within the space budget, the relatively "important" QFRs (those that refine parts of the histogram that are not refined by other QFRs) must be identified and retained, and the less important QFRs discarded.

ISOMER addresses each of the above challenges using novel, efficient techniques:

1. ISOMER uses the maximum-entropy principle (Section 3.3) to approximate the true data distribution by the "simplest" distribution that is consistent with all of the currently valid QFRs. This approach amounts to imposing uniformity assumptions (as made by traditional optimizers) when, and only when, no other statistical information is available. ISOMER efficiently updates the maximum-entropy approximation in an incremental manner as new QFRs arrive.

2. ISOMER employs linear programming (LP) to quickly detect and discard old, invalid QFRs.

3. An elegant feature of ISOMER is that the maximum-entropy solution yields, for free, an "importance" measure for each QFR. Thus, to meet a limited space budget, ISOMER simply needs to discard QFRs in increasing order of this importance measure.

Our experiments indicate that for reactive histogram maintenance, ISOMER is highly efficient, imposes very little overhead during query execution, and can provide much more accurate cardinality estimates than previous techniques while using only a modest amount of storage.

The rest of the paper is organized as follows. After surveying related work in the remainder of this section, we lay out the basic architecture of ISOMER in Section 2 and introduce its various components. In Section 3, we describe how ISOMER uses the maximum-entropy principle to build histograms consistent with query feedback. Section 4 describes how ISOMER deals with changing data, and Section 5 details how ISOMER keeps the histogram within a limited space budget by discarding relatively unimportant QFRs. We describe our experimental results in Section 6, and conclude in Section 7.

1.1. Related Work

Existing work on multidimensional statistics can be broadly classified as addressing either the problem of deciding which statistics to build or that of actually building them. This paper addresses only the latter problem. Many types of statistics have been proposed, e.g., histograms [15] and wavelet-based synopses [13]; we restrict attention to histograms.

For building multidimensional histograms, proactive approaches that involve a data scan have been proposed, e.g., MHist [15], GenHist [9], and others [7, 14, 19]. As mentioned before, data scans may not effectively focus system resources on the user's workload and do not scale well to large tables. In principle, histograms can be constructed faster using a page-level sample of the data [5], but large sample sizes—and correspondingly high sampling costs—can be required to achieve sufficient accuracy when data values are clustered on pages and/or highly skewed.

The idea of using query feedback to collect the statistics needed for estimating cardinality was first proposed in [6]. The specific approach relied on fitting a combination of model functions to the data distribution; the choice of functions is *ad hoc* and can lead to poor estimates when the data distribution is irregular. Query feedback is also used in [17], but only to compute adjustment factors to cardinality estimates for specific predicates, and not to build histograms. STGrid [1] and SASH [11] both use query feedback to build histograms, but they often have low accuracy because their heuristic methods for adding new QFRs to the histogram do not maintain consistency. ISOMER's use of the well founded [16] maximum-entropy principle avoids this problem.

STHoles [4] is another approach that uses query feedback to build histograms. The histogram structure of STHoles is superior to other bucketing schemes such as

MHist [15], and for this reason is used by ISOMER. Unfortunately, the original STHoles maintenance algorithm requires, for each query and each histogram bucket, the computation of the number of rows in the intersection of the query and bucket regions. These detailed row counts, which are used to decide when and where to split and merge buckets, are usually not obtainable from the original query predicates alone. The query engine must therefore insert artificial predicates that specify the (possibly recursive) bucket boundaries. As the number of histograms and the number of buckets per histogram grows, the overhead of evaluating this "artificial feedback" becomes so high as to make the STHoles maintenance approach impractical. (ISOMER, in contrast, needs only the actual feedback that naturally occurs during query execution—namely, the number of rows processed at each step during the query plan—which can be monitored with low overhead [17].) Finally, STHoles, unlike ISOMER, does not provide principled methods for addressing issues of inconsistent feedback and limits on the available memory for storing the histogram.

The principle of maximum entropy has been used in [12], but for the significantly different problem of consistently estimating the selectivity of conjuncts of predicates such as $\text{sel}(p_1 \wedge p_2 \wedge p_3 \wedge p_4)$, given partial selectivities such as $\text{sel}(p_1)$, $\text{sel}(p_2 \wedge p_3)$, $\text{sel}(p_2 \wedge p_3 \wedge p_4)$, and so forth. I.e., the methods in [12] permit the exploitation of existing multidimensional statistics (not necessarily from histograms), whereas the current paper is concerned with the collection of a specific type of statistic.

2. ISOMER architecture

Figure 1 shows the architecture of ISOMER and where it would fit in a modern commercial DBMS. The left portion of the figure shows an ordinary DBMS. Here, a query is first fed to a cost-based query optimizer, which produces a plan for the query based on the available database statistics. Presently, all commercial systems apply the proactive approach for maintaining statistics that requires database access (as shown by the dotted arrow). The chosen query plan is then executed by the runtime engine. We prototyped ISOMER on top of DB2 UDB, which can measure and store actual predicate cardinalities or QFRs at run time.

Due to efficiency concerns, a QFR is not immediately used to refine the histogram as soon as it is obtained. Instead, the QFRs are collected in a query feedback store and used in batches to refine the histogram. During periods of light load, or during a maintenance window, the database systems invokes the ISOMER component. On invocation, ISOMER uses the QFRs accumulated in the query feedback store to refine the histogram. The right portion of the figure shows the various components of ISOMER.

ISOMER begins by reading the currently maintained histogram from the database statistics. It then accesses the new QFRs and adds them to the current histogram. This process mainly involves forming buckets corresponding to the new QFRs, and is explained in Section 3.4.1.

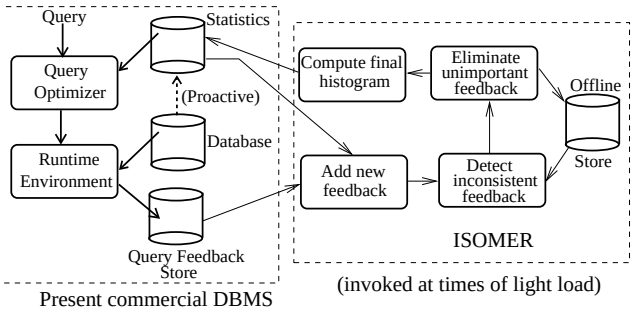

Figure 1. ISOMER Architecture

As discussed in Section 3.2 below, a histogram must be consistent with respect to both new and previous QFRs. However, because of updates to the data these QFRs may be contradictory, in which case there does not exist a histogram consistent with all QFRs. Thus ISOMER's first task is to detect and discard old, inconsistent QFRs. For this purpose, ISOMER keeps a list of QFRs previously added to the histogram in an *offline store* as depicted in Figure 1. ISOMER reads the previous QFRs from the offline store and uses linear programming to find and eliminate inconsistent QFRs. This process is described in Section 4. Note that the offline store does not have to be read at query-optimization time and hence does not encroach on the space budget allowed for the histogram. In any case, the size of the offline store is not a concern since it cannot grow bigger than the size of the maintained histogram. Indeed, if there are more QFRs than buckets, then some of these QFRs will be either inconsistent or redundant and hence will be eliminated; see Section 3.4.2.

Once ISOMER obtains a consistent set of QFRs, the algorithm computes the histogram according to the maximum-entropy principle. We describe this computation in Sections 3.4.2 and 3.4.3. If the histogram is too large to fit within its allotted space, ISOMER selectively discards the relatively "unimportant" QFRs in order to reduce the histogram size. Intuitively, a particular QFR is unimportant if the information provided by that QFR is already provided by other QFRs, i.e., if the QFR refines a portion of the histogram that is already sufficiently refined by other QFRs. The process for detecting and discarding unimportant QFRs is described in Section 5.

Any inconsistent or unimportant QFR that ISOMER discards is also removed from the list of QFRs that are currently incorporated into the histogram, and the revised list is written back to the offline store. The final histogram is computed according to the maximum-entropy principle and written back into the database statistics.

3. Incorporating Feedback

After defining the notions of histogram and query feedback that we use throughout the paper (Section 3.1), we introduce a running example (Section 3.2) that illustrates the use of query feedback to build and refine a histogram, the importance of maintaining consistency, and the difficulties that arise therein. We then introduce the maximum-entropy principle to address these issues (Section 3.3), and explain its use in ISOMER (Section 3.4).

3.1. Histograms and Feedback

Given a table T comprising N tuples, we wish to build a d-dimensional histogram over attributes $A_1, \ldots, A_d$ of table T. For each numerical attribute A_i, denote by l_i and u_i the minimum and maximum values of A_i in the table. We assume that these values are available from one-dimensional database statistics on A_i. A categorical attribute A_i having D_i distinct values can be treated as numerical by mapping each distinct value to a unique integer in $[1, D_i]$, so that $l_i = 1$ and $u_i = D_i$. The space in which the tuple values lie is $\mathscr{S} = [l_1, u_1] \times [l_2, u_2] \times \ldots \times [l_d, u_d]$.

A multidimensional *histogram* is a lossy compressed representation of the true distribution of tuples in $\mathscr{S}$ that is obtained by partitioning $\mathscr{S}$ into $k \geq 1$ mutually disjoint regions called *buckets*, and recording the number of tuples in T that fall in each bucket. ISOMER actually maintains an approximate histogram that records an estimate of the number of tuples that fall in each bucket. Denote by $b_1, b_2, \ldots, b_k$ the k histogram buckets, by $C(b_i)$ the region of $\mathscr{S}$ that is covered by b_i, and by $n(b_i)$ the number of tuples estimated to lie in b_i. The tuples in each bucket b_i are assumed to be uniformly distributed throughout $C(b_i)$.

ISOMER maintains a histogram based on *query feedback*. Specifically, at each time point, the system maintains a list of QFRs of the form $\big(q_1, N(q_1)\big), \big(q_2, N(q_2)\big), \ldots, \big(q_m, N(q_m)\big)$ for some $m \geq 1$, where each q is a predicate of the form[1]

$$(x_1 \leq C_1 \leq y_1) \wedge \cdots \wedge (x_{d'} \leq C_{d'} \leq y_{d'})$$

and $N(q)$ is the number of tuples that satisfy q. Here $d' \leq d$ and $C_1, \ldots, C_{d'}$ are distinct attributes from among $A_1, \ldots, A_d$. We assume that if C_j is categorical, then $x_j = y_j$, so that the conjunct $x_j \leq C_j \leq y_j$ is actually an equality predicate. In the following sections, we denote by $R(q)$ the subset of the region $\mathscr{S}$ for which q is true.

3.2. A Running Example

Consider a `Car` relation with attributes `make` and `color`, and suppose that a user executes the query

```
SELECT * FROM Car WHERE
make = 'Honda' AND color = 'White'
```

[1]The maximum-entropy principle can be applied to general predicates, but we have initially focused on conjunctive predicates in ISOMER.

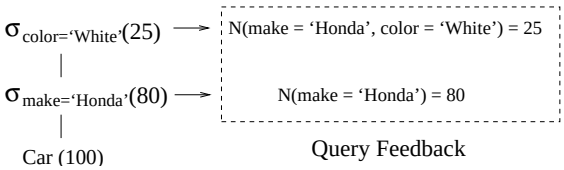

Figure 2. Gathering Query Feedback (QFRs)

Figure 2 shows a possible plan for executing this query; the actual number of tuples at each stage are shown in parenthesis. The figure also shows the various predicates q for which $N(q)$ can be collected during query execution; the cardinality information for these predicates comprise the QFRs. Gathering of QFRs is already supported by several commercial database systems and has been shown [17] to have very low runtime overhead (typically less than 5%).

Suppose that we wish to build a two-dimensional histogram on the attributes `make` and `color` by using query feedback. Also suppose for simplicity that there are only two distinct makes (say `'Honda'` and `'BMW'`), and only two distinct colors (say `'Black'` and `'White'`) in the `Car` relation. Then any multidimensional histogram on these attributes has at most four buckets (Figure 3). We assume that our histogram stores all of these four buckets.

The first QFR that we add to the histogram is the total number of tuples in the table. This number can be obtained from system catalog statistics. Suppose there are 100 tuples in the `Car` table. This QFR can be expressed as:

$$n(b_1) + n(b_2) + n(b_3) + n(b_4) = 100 \tag{1}$$

At this point, there are various possible assignments of values to the $n(b_i)$'s that will make the histogram consistent with this single QFR. In the absence of any additional knowledge, we assume, as do traditional optimizers, that values are distributed uniformly. Hence, the histogram obtained after adding this QFR is one in which each $n(b_i)$ equals 25, as shown in Figure 3(b).

We now add the QFR $N($`make = 'Honda'`$) = 80$ to the histogram. This QFR can be expressed as:

$$n(b_2) + n(b_4) = 80 \tag{2}$$

To make the histogram consistent with (2) while preserving uniformity between $n(b_2)$ and $n(b_4)$, we set $n(b_2) = n(b_4) = 40$. The resulting histogram is shown in Figure 3(c). At this point, the STGrid [1] approach would consider the process of adding this QFR to be finished, with Figure 3(c) as the final histogram.

Notice, however, that the histogram in Figure 3(c) is no longer consistent with (1). The expressions in (1) and (2) together imply that $n(b_1) + n(b_3) = 20$. To enforce consistency with this equation as well as uniformity between $n(b_1)$ and $n(b_3)$, we set $n(b_1) = n(b_3) = 10$. The final consistent histogram is shown in Figure 3(d).

Observe that even though we added information only about Hondas, the histogram now gives a more accurate es-

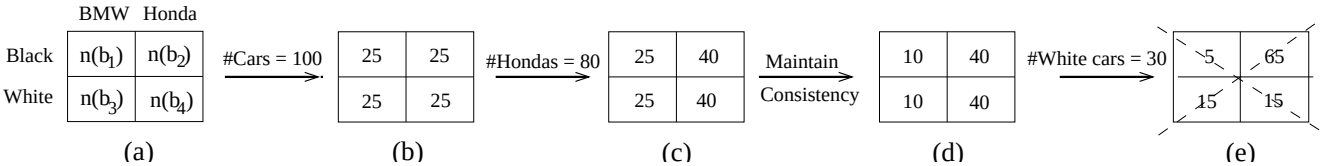

Figure 3. Running Example

timate of the frequency of BMWs. This improvement is a direct result of our final histogram adjustment in which we enforced consistency with both the QFRs.

In summary, it is reasonable to impose the following requirements when adding a new QFR to the histogram:

1. **Consistency**: After adding a new QFR, the histogram should be consistent with all QFRs added so far.
2. **Uniformity**: If there are multiple histograms consistent with all of the QFRs, then the choice of the final histogram should not be arbitrary, but must be based on the traditional assumption of uniformity.

So far in this example, it has been easy and intuitive to apply the uniformity assumption. However, it is not always clear how to impose uniformity. To illustrate, suppose that the next QFR we add to the histogram is $N(\texttt{color = 'white'}) = 30$. This QFR can be written as:

$$n(b_3) + n(b_4) = 30. \tag{3}$$

If we employ the naïve solution of making the histogram consistent with (3) while imposing uniformity on $n(b_3)$ and $n(b_4)$, we get $n(b_3) = n(b_4) = 15$. Further enforcing consistency with (1) and (2), we get the final histogram shown in Figure 3(e). This solution clearly does not enforce uniformity[2]: although white Hondas and white BMWs are in equal proportion, there are far more black Hondas than black BMWs. This non-uniformity among black cars has not been indicated by any added QFR, and is only due to our ad-hoc method of enforcing consistency. To enforce consistency while avoiding unsatisfactory results as in Figure 3(e), we turn to the information-theoretic principle of maximum entropy, described next.

3.3. The Maximum-Entropy Principle

Consider a discrete probability distribution $(p_1, p_2, \ldots, p_n)$ on n distinct outcomes, i.e., each p_i is nonnegative and $\sum_{i=1}^{n} p_i = 1$. In many applications, only partial information about such a probability distribution is available (e.g., $p_1 + p_2 = 0.5$). Define a "candidate" distribution as one that is consistent with all available information about the distribution. In general, there may be multiple candidate distributions. The maximum-entropy principle (see, e.g., [16]) provides a well grounded criterion for selecting a unique distribution from among the candidates. Specifically, the principle prescribes selection of the candidate $\mathscr{P} = (p_1, p_2, \ldots, p_n)$ that has the maximum entropy value $H(\mathscr{P})$, where $H(\mathscr{P}) = -\sum_{i=1}^{n} p_i \ln(p_i)$. The maximum-entropy principle can be justified informally as follows. From information theory, we know that entropy measures the uncertainty or uninformativeness in a distribution. For example, the value of the entropy ranges from 0—when a specified outcome occurs with certainty—to a maximum of $\ln(n)$ when no information is available and all outcomes are equally likely ($p_1 = \cdots = p_n = 1/n$). Thus the maximum-entropy principle leads to a choice of the simplest, i.e., most uninformative distribution possible that is consistent with the available information. To choose a distribution with lower entropy would amount to assuming information that we do not have; to choose one with a higher entropy would ignore the information that we do have. The maximum-entropy distribution is therefore the only reasonable choice. A more formal justification of the principle can be found in [16]. For a continuous probability distribution with probability density function (pdf) $\mathscr{P}(u)$, the entropy is defined as $H(\mathscr{P}) = \int_{\mathscr{S}} \mathscr{P}(u) \ln\big(\mathscr{P}(u)\big)\, du$, and the foregoing discussion extends to this setting in an obvious way.

For ISOMER, the maximum-entropy principle can also be justified as a means of ensuring uniformity. As mentioned above, entropy is maximized for a uniform distribution. Thus, choosing the distribution according to the maximum-entropy principle facilitates our goal of maintaining consistency and uniformity at the same time.

3.4. Maximum Entropy in ISOMER

We first describe the histogram structure used in ISOMER (Section 3.4.1). We then show in Section 3.4.2 how application of the maximum-entropy principle leads to an optimization problem. Section 3.4.3 describes how this optimization problem is solved in ISOMER.

3.4.1 Histogram Structure in ISOMER

ISOMER uses the STHoles data structure [4] to represent and store multidimensional histograms. In the STHoles histogram, each bucket b_i has a hyperrectangular bounding box denoted by $\text{box}(b_i)$ ($\subseteq \mathscr{S}$), i.e., bucket b_i is bounded between two constant values in each dimension. Bucket b_i, however, does not cover the entire region $\text{box}(b_i)$. There

[2] STHoles [4] does not face this problem of enforcing uniformity because for a predicate q, it explicitly gathers a count of the intersection of $R(q)$ with every histogram bucket. Thus, STHoles would gather $n(b_3)$ and $n(b_4)$ individually, making it a high-overhead approach in general.

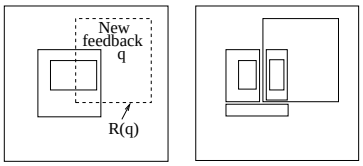

Figure 4. Drilling Holes for a New QFR

may be some "holes" inside box(b_i) that are not covered by b_i. These regions are themselves histogram buckets, and are referred to as *children* of b_i. The bounding boxes of these children are mutually disjoint hyperrectangles and completely enclosed within box(b_i). The region covered by b_i is formally given by:

$$C(b_i) = \text{box}(b_i) - \bigcup_{b_j \in children(b_i)} \text{box}(b_j)$$

Intuitively, in the absence of holes, b_i would represent a region of uniform tuple density. However, the STHoles histogram identifies regions within b_i that have a different tuple density, and represents them as separate histogram buckets.

For a multidimensional predicate q that selects all tuples lying in a specified region $R(q) \subseteq \mathscr{S}$, an STHoles histogram comprising buckets $b_1, b_2, \ldots, b_k$ estimates the number of tuples that satisfy this predicate as

$$\hat{N}(q) = \sum_{i=1}^{k} n(b_i) \frac{\text{vol}\big(R(q) \cap C(b_i)\big)}{\text{vol}\big(C(b_i)\big)}. \tag{4}$$

Here vol(R) denotes the usual euclidean volume of the region R when the data is real-valued; for discrete (i.e., integer or integer-coded categorical) data, vol(R) denotes the number of integer points that lie in R.

ISOMER initializes the STHoles histogram to contain a single bucket b_1 such that box$(b_1) = C(b_1) = \mathscr{S}$ and $n(b_1) = N$. At this point, the histogram embodies the simplest possible uniformity assumption. As more QFRs are added over time, ISOMER learns more about the distribution of tuples in $\mathscr{S}$ and incorporates this information into the histogram by "drilling" holes in b_1.

ISOMER's technique for drilling holes is a simpler version of the method given in [4]. Suppose that ISOMER obtains a QFR about a specified multidimensional predicate q. To make the histogram consistent with this QFR, ISOMER must first ensure that the histogram contains a set of buckets that exactly cover $R(q)$, so that the sum of the tuple counts in these buckets can then be equated to $N(q)$ as in Section 3.2. If such a set of buckets already exists, no holes need to be drilled. Otherwise, the process of drilling holes for q proceeds as shown in Figure 4. Specifically, ISOMER descends down the bucket tree until it finds a bucket b such that $R(q) \subset C(b)$ but $R(q) \not\subseteq C(b')$ for any $b' \in$ children(b). ISOMER forms a new bucket b_{new} such that box$(b_{new}) = R(q)$ and processes each bucket $b' \in$ children(b) as follows.

- If box$(b') \cap R(q) = \emptyset$, then nothing needs to be done.
- If box$(b') \subset R(q)$, then b' is removed from children(b) and added to children(b_{new}).
- If box(b') partially overlaps $R(q)$, then bucket b' (and recursively its children), are split as shown in Figure 4 to preserve disjointness and a hyperrectangular shape. The splitting is done one dimension at a time, using an arbitrary ordering among the dimensions.

3.4.2 Formulation of the Optimization Problem

To apply the maximum-entropy principle in ISOMER, we associate a probability distribution $\mathscr{P}$, and hence an entropy value $H(\mathscr{P})$, with every possible histogram. In accordance with the maximum-entropy principle, ISOMER then maximizes $H(\mathscr{P})$ over the set of all histograms that are consistent with the current set of QFRs. To define $\mathscr{P}$ and $H(\mathscr{P})$, consider an STHoles histogram with buckets $b_1, \ldots, b_k$ having bucket counts $n(b_1), \ldots, n(b_k)$. If the data is discrete, then $\mathscr{P}$ is a probability distribution over the integer points of $\mathscr{S}$ given by $p_u = n(b_u^*)/[N \cdot V(b_u^*)]$ for $u \in \mathscr{S}$, where $V(b)$ is abbreviated notation for vol$\big(C(b)\big)$, and b_u^* is the unique bucket b such that $u \in C(b)$. This definition follows from (4) after dividing both sides by the total number of tuples N and taking q to be the point query "$(A_1, A_2, \ldots, A_d) = u$". The entropy $H(\mathscr{P}) = -\sum_{u \in \mathscr{S}} p_u \ln(p_u)$ corresponding to the distribution $\mathscr{P}$ is thus given by

$$\begin{aligned} H(\mathscr{P}) &= -\sum_{u \in \mathscr{S}} \frac{n(b_u^*)}{N \cdot V(b_u^*)} \ln\left(\frac{n(b_u^*)}{N \cdot V(b_u^*)}\right) \\ &= -\sum_{i=1}^{k} \sum_{u \in C(b_i)} \frac{n(b_i)}{N \cdot V(b_i)} \ln\left(\frac{n(b_i)}{N \cdot V(b_i)}\right). \end{aligned}$$

Since the inner sum comprises $V(b_i)$ identical terms that are independent of u,

$$\begin{aligned} H(\mathscr{P}) &= -\sum_{i=1}^{k} \frac{n(b_i)}{N} \ln\left(\frac{n(b_i)}{N \cdot V(b_i)}\right) \\ &= -\frac{1}{N} \sum_{i=1}^{k} n(b_i) \ln\left(\frac{n(b_i)}{V(b_i)}\right) + \ln(N), \end{aligned} \tag{5}$$

where the last equality uses the identity $\sum_{i=1}^{k} n(b_i) = N$. For real-valued data, we take $\mathscr{P}$ to be the pdf defined by $\mathscr{P}(u) = p_u$ for each real-valued point $u \in \mathscr{S}$, where p_u is defined as above; note that this density function is constant within each region $C(b_i)$. A straightforward calculation shows that the entropy $H(\mathscr{P}) = \int_{\mathscr{S}} \mathscr{P}(u) \ln\big(\mathscr{P}(u)\big)\, du$ is given by (5).

We now express the QFRs as constraints on the histogram. Suppose that ISOMER has obtained QFRs for m predicates $q_1, \ldots, q_m$. First, ISOMER drills holes for these QFRs in the histogram as described in Section 3.4.1. For each q_i, the drilling procedure ensures that the set of histogram buckets lying within $R(q_i)$ exactly cover $R(q_i)$. Hence the QFR for q_i can be written as the constraint

$$\sum_{b | C(b) \subseteq R(q_i)} n(b) = N(q_i) \tag{6}$$

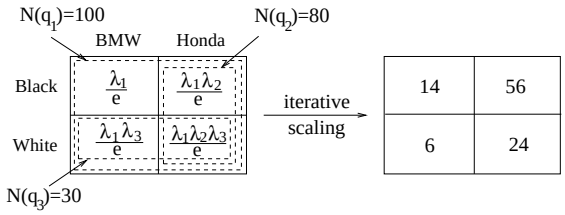

Figure 5. Iterative Scaling Example

The application of the maximum-entropy principle thus leads to a well posed[3] optimization problem: for an STHoles histogram with buckets $b_1, \ldots, b_k$, select nonnegative bucket counts $n(b_1), \ldots, n(b_k)$ so as to maximize the expression $H(\mathscr{P})$ in (5), while satisfying (6) for $1 \leq i \leq m$.

3.4.3 Solution of Optimization Problem

To solve the above optimization problem, associate a Lagrange multiplier y_i ($1 \leq i \leq m$) with the ith constraint given by (6). After removing the constants from the objective function in (5), the Lagrangian of the optimization problem is given by:

$$\sum_{i=1}^{m} y_i \left(\sum_{b|C(b)\subseteq R(q_i)} n(b) - N(q_i) \right) - \sum_{i=1}^{k} n(b_i) \ln \left(\frac{n(b_i)}{V(b_i)} \right)$$

Differentiate the above expression with respect to $n(b)$ and equate to 0 to get

$$\sum_{i|C(b)\subseteq R(q_i)} y_i - \ln \left(\frac{n(b)}{V(b)} \right) - 1 = 0,$$

so that, setting $\lambda_i = e^{y_i}$,

$$n(b) = \frac{V(b)}{e} \prod_{i|C(b)\subseteq R(q_i)} \lambda_i. \tag{7}$$

Combining (7) with (6), we get a system of m equations, where there are m unknowns $\lambda_1, \ldots, \lambda_m$. This system of equations can be solved by a procedure known as iterative scaling. The details of iterative scaling are omitted due to lack of space and can be found in [12].[4]

Example 1 *For the example of Section 3.2, $V(b_i) = 1$ for each i. As before, we add three QFRs given by* (1), (2), *and* (3). *Denote these QFRs as q_1, q_2, and q_3, respectively; see Figure 5. Since bucket b_1 is referred to only by q_1, it follows from* (7) *that $n(b_1) = \lambda_1/e$. Similarly, $n(b_2) = \lambda_1\lambda_2/e$, $n(b_3) = \lambda_1\lambda_3/e$, and $n(b_4) = \lambda_1\lambda_2\lambda_3/e$. We can therefore rewrite* (1), (2), *and* (3) *as*

$$\begin{aligned} \lambda_1 + \lambda_1\lambda_2 + \lambda_1\lambda_3 + \lambda_1\lambda_2\lambda_3 &= 100e \\ \lambda_1\lambda_2 + \lambda_1\lambda_2\lambda_3 &= 80e \\ \lambda_1\lambda_3 + \lambda_1\lambda_2\lambda_3 &= 30e \end{aligned}$$

Solving the above equations, we obtain $\lambda_1 = 14e$, $\lambda_2 = 4$, and $\lambda_3 = 3/7$, yielding the final histogram (Figure 5).

This histogram is consistent with all of the added QFRs. It is also "the most uniform" in the following sense. It maintains the 80-20 ratio between Hondas and BMWs for both the colors. Similarly, it maintains the 30-70 ratio between white and black cars for both the makes. Such uniformity is not obtained by adding QFRs in an ad-hoc manner, e.g., as in the histogram of Figure 3(e).

4. Dealing with Database Updates

As long as no tuples are updated, inserted, or deleted, all of the QFRs obtained by ISOMER are consistent with each other, and there exists at least one valid histogram solution that satisfies the set of constraints in (6). However, in the presence of data changes, the set of QFRs might evolve to a point at which no histogram can be simultaneously consistent with the entire set.

Example 2 *Suppose that ISOMER obtains the two QFRs $N($*`make = 'Honda'`$) = 80$ *and $N($*`make = 'Honda',color = 'White'`$) = 30$, and then some updates to the data occur. After these updates, ISOMER might obtain the QFR $N($*`make = 'Honda', color = 'Black'`$) = 60$. Clearly, there exists no histogram solution consistent with all three QFRs.*

Given inconsistent QFRs, there exists no solution to the optimization problem of Section 3.4.2. Thus, ISOMER must first discard the QFRs that are no longer valid due to data changes, leaving a set of consistent QFRs having a valid histogram solution. However, deciding which QFR is invalid is not always straightforward and depends on the type of data change. In Example 2, if some new black-Honda tuples have been inserted, the first QFR is invalid. However, if the color of some Hondas has been updated from white to black, then the second QFR is invalid. In general, both the first and second QFRs may be invalid.

Since no information is available about the type of data change, ISOMER uses the notion of the *age* of a QFR to decide which QFRs to discard. The intuition is that the older a QFR, the more likely that it has been invalidated. Thus ISOMER discards those QFRs that are relatively old and whose removal leaves behind a consistent set. To quickly detect such QFRs, the following LP approach is employed.

[3] There may be some buckets b for which $n(b) = 0$ in any consistent solution. We detect such buckets by solving an appropriate linear program, and exclude them from the entropy calculation since their contribution to the entropy is 0 ($\lim_{n(b)\to 0} n(b) \ln\big(n(b)\big) = 0$).

[4] We actually modify the basic algorithm in [12] to permit efficient incremental updating of the maximum entropy solution. The idea is to persist the multipliers corresponding to each QFR in the offline store, so that we only need to recompute the solution for a few specified regions. Details are omitted for brevity.

4.1. A Linear-Programming Solution

ISOMER associates two "slack" variables with each constraint corresponding to the QFRs. The constraints in (6) are rewritten as (for $1 \leq i \leq m$)

$$\sum_{b|C(b)\subseteq R(q_i)} n(b) - N(q_i) = s_i^+ - s_i^- \tag{8}$$

ISOMER also adds the nonnegativity constraints

$$n(b) \geq 0 \text{ for all } b, \qquad s_i^+, s_i^- \geq 0 \text{ for } i = 1, \ldots, m. \tag{9}$$

If there is a solution to the set of constraints (8) and (9) such that $s_i^+ = s_i^- = 0$, then the solution satisfies the ith constraint from (6). Otherwise, if s_i^+ or s_i^- is positive, the ith constraint is not satisfied. Ideally, we would like a solution that satisfies the maximum number of constraints from (6), i.e., a solution that minimizes the number of nonzero slack variables. Unfortunately, determining such a solution is known to be NP-complete [2]. ISOMER instead settles for minimizing the sum of the slack variables, because this problem can be solved by linear programming. ISOMER then discards all QFRs that correspond to constraints having nonzero slack. Note that if all the original constraints from (6) are satisfiable, then there exists a solution in which all of the slacks equal 0, and hence no QFRs are discarded.

As noted earlier, we want to preferentially discard older QFRs. Instead of minimizing the sum of slack variables, ISOMER therefore minimizes a weighted sum of the slack variables, where the slack corresponding to a QFR is weighted inversely by the age of the QFR. Thus, a QFR that is not satisfied and has nonzero slack incurs a smaller objective-function penalty if it is old than if it is new. Thus an optimal solution is more likely to permit slack in older QFRs, so that such QFRs are preferentially discarded. The age of the ith QFR is given by $m - i + 1$. Thus ISOMER solves the following linear program to detect inconsistent constraints: Minimize

$$\sum_{i=1}^{m} \frac{1}{m-i+1}(s_i^+ + s_i^-)$$

subject to (8) and (9). If s_i^+ or s_i^- is nonzero in the resulting solution, then the ith QFR is discarded. In our implementation, we have used the highly optimized open source Coin LP solver [18]. Discarding QFRs enables ISOMER to merge buckets as described in the next subsection.

Our overall method for eliminating invalidated QFRs is clearly a heuristic. However, as shown in Section 6, it works effectively and efficiently in practice. In future work, we hope to enhance ISOMER with a more principled method for eliminating inconsistent QFRs.

4.2. Merging Histogram Buckets

Once ISOMER has decided to discard a particular QFR, the total histogram size can potentially be reduced by merging two or more histogram buckets. The process of merging buckets is essentially the inverse of the process for drilling holes described in Section 3.4.1 and is similar to that described in [4]. Due to space constraints, we give only a brief description here.

After the ith QFR is discarded, ISOMER examines all "top-level" buckets that cover $R(q_i)$, i.e., all buckets b such that $C(b) \subseteq R(q_i)$, but $C\big(\text{parent}(b)\big) \nsubseteq R(q_i)$. Bucket b can be merged with another bucket in the following cases:

- If bucket b has exactly the same referring set as its parent, i.e., if $\{j|C(b) \subseteq R(q_j)\} = \{j|C\big(\text{parent}(b)\big) \subseteq R(q_j)\}$, then b can be merged with its parent. This is because both b and parent(b) have the same number of tuples per unit volume in the maximum-entropy solution; cf. (7). The children of b now become the children of parent(b).
- If there is a sibling b' of b such that (i) b' and b have the same referring set and (ii) box$(b') \cup$ box(b) is hyperrectangular, then b and b' can be merged. The children of b are recursively examined to see if they can be merged with their new siblings, i.e., children of b'. The new merged bucket is also examined to see if it can be merged with any of its siblings.

5. Meeting a Space Budget

Database systems usually limit the size of a histogram in order to ensure efficiency when the histogram is read at optimization time. Thus, we assume that there is limited space available for histogram storage. The addition of new QFRs causes the size of ISOMER's STHoles histogram to grow as new holes are drilled. Whenever the histogram size exceeds the space budget, ISOMER reduces the histogram size by discarding "unimportant" QFRs. Intuitively, a QFR is unimportant if it provides little information over and above what is already provided by other QFRs. Discarding unimportant QFRs reduces the total histogram size by triggering the merging of buckets (Section 4.2).

Example 3 *In the example of Section 3.2, suppose that we have space to store only two histogram buckets. After adding the QFRs* $N = 100$ *and* $N($`make = 'Honda'`$) = 80$*, the resulting histogram has two buckets, and is shown in Figure 6(a). Each bucket has volume equal to 2, and bucket* b_2 *is a hole in the top-level bucket* b_1*. Suppose that we now add the third QFR,* $N($`make = 'BMW', color = 'white'`$) = 10$*. ISOMER drills a hole corresponding to this QFR, and the resulting histogram, shown in Figure 6(b), has three buckets, violating the space budget. Notice, however, that the addition of this third QFR yields no extra information: because tuples are assumed to be uniformly distributed within a bucket, the histogram in Figure 6(b) is already implied by the histogram in Figure 6(a). Thus the third QFR is unimportant, and can be discarded.*

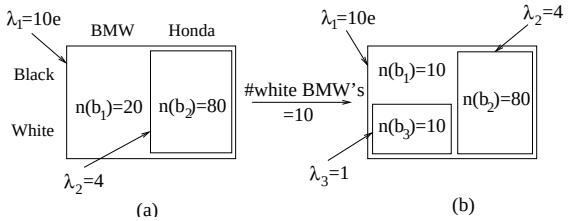

Figure 6. Unimportant QFR Example

How can the unimportant QFRs be efficiently determined? Note that the age of a QFR—which ISOMER uses as a criteria for deciding which QFRs are invalid—is not relevant for deciding importance. E.g., in Example 3, ISOMER can receive many instances of the third QFR in succession, thus making the second QFR very old. However, the second QFR is still more important than the third QFR.

An elegant aspect of ISOMER is that the maximum-entropy solution yields, for free, an importance measure for each QFR. This leads to a very efficient procedure for detecting and discarding unimportant QFRs. Specifically, ISOMER uses the quantity $|\ln(\lambda_i)|$ as the importance measure for the ith QFR, where λ_i is the multiplier defined in (7). To justify this choice intuitively, we note that if $\lambda_i = 1$, then removal of the ith QFR does not affect the bucket counts, so that the final maximum-entropy solution is unaltered. For instance, $\lambda_3 = 1$ in the final solution in Example 3—see Figure 6(b)—and we concluded that the third QFR was unimportant. By the same token, if the multiplier corresponding to a QFR is close to 1, then removal of that QFR will affect the final solution less than if we remove a QFR whose multiplier is much greater than 1. Thus, the ith QFR is unimportant if $\lambda_i \approx 1$, i.e., if $|\ln(\lambda_i)| \approx 0$.

An alternative justification for our definition follows from the fact that $|\ln(\lambda_i)| = |y_i|$, where y_i is the Lagrange multiplier corresponding to the ith constraint in the maximum-entropy optimization problem (Section 3.4.3). It is well known from optimization theory [3] that the Lagrange multiplier for a constraint measures the degree to which the constraint affects the optimum value of the objective function. Thus $|y_i|$ measures how much the ith constraint affects the entropy, i.e., the amount of information in the distribution. In other words, $|y_i| = |\ln(\lambda_i)|$ is a measure of the amount of information carried by the ith constraint, and hence a measure of the importance of the ith QFR.

Thus, whenever the histogram exceeds the space budget, ISOMER proceeds by examining the current maximum-entropy solution and, for each i, computes the importance measure for the ith QFR as $|\ln(\lambda_i)|$. ISOMER then discards the QFR with the least importance according to this measure and merges buckets as described in Section 4.2. ISOMER then incrementally computes the new maximum-entropy solution and repeats the above procedure until the histogram is sufficiently small.

6. Experiments

In this section, we provide an experimental validation of ISOMER. We prototyped ISOMER on top of DB2 UDB, which supports gathering of query feedback. The experiments were conducted on an Intel 2.3GHz processor with 1GB RAM. We used ISOMER to maintain a multidimensional histogram using query feedback and compared ISOMER's cardinality estimates against estimates obtained by a state-of-the-art commercial optimizer. The only form of multidimensional statistics presently supported by the optimizer is a count of the number of distinct values in a group of columns. We also compared the ISOMER estimates with those obtained by STGrid, since both algorithms use the same type of feedback. We did not compare ISOMER with STHoles, because the feedback collection strategy used by STHoles has a very high overhead that makes the approach impractical, particularly when index scans are used during query processing; see Section 1.1.

Our experiments demonstrate the following:

1. ISOMER provides significantly better cardinality estimates than either the commercial optimizer technique or STGrid.
2. For static data, the cardinality estimates provided by ISOMER consistently improve and then stabilize as QFRs are added to the histogram, i.e., there is no oscillation in accuracy when more QFRs are added (and others removed to keep within the space budget).
3. For changing data, ISOMER learns the new data distribution much faster (i.e., with the addition of many fewer QFRs) than STGrid and provides significantly better cardinality estimates.
4. The overall overhead of ISOMER is low.

For our experiments we used a DMV (Department of Motor Vehicles) database that was derived from a real-world application. For our experiment, we focused on the `Cars` table which has several correlated columns such as `make`, `model`, `color`, and `year`. The maximum number of distinct values in any column was approximately 150.

We generated a collection of queries referred to as training queries, issued them to the query execution engine, and collected QFRs for the predicates in these queries. We used ISOMER (or STGrid for comparison) to maintain a multidimensional histogram on the set of referenced attributes. The histogram was initialized using one-dimensional database statistics on the attributes and by assuming independence between the attributes. QFRs collected during execution of the training queries were then used to refine this histogram while allowing a maximum of k buckets, where k was an experimental parameter.

To test the accuracy of the maintained histogram, we generated a collection of 200 test queries from the same

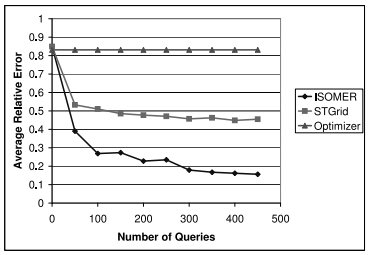

Figure 7. Error of 2D Histogram on `model` and `year`

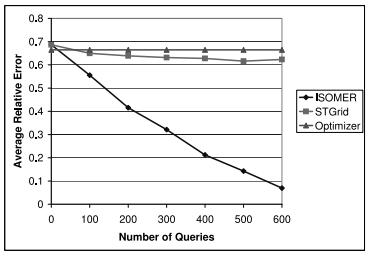

Figure 8. Error of 3D Histogram on `make`, `model` and `color`

distribution as that of the training queries. We periodically tested the histogram's accuracy by comparing the actual and estimated cardinalities for each predicate in the test queries. We measured the relative error in the estimation as

$$\text{Relative Error} = \frac{|N(q) - \hat{N}(q)|}{\max(100, N(q))},$$

where $N(q)$ and $\hat{N}(q)$ are the actual and estimated number of tuples respectively that satisfy the predicate q. In our formula, the quantity 100 is a sanity constant [8] that prevents the relative error from blowing up in case of highly selective predicates, i.e., predicates for which $N(q)$ is either very small or equal to 0. We measured the overall accuracy of the histogram by the average relative error across all predicates in the test queries.

We discuss the accuracy of ISOMER on static data in Section 6.1 and then consider updates in Section 6.2. We study the running time of ISOMER in Section 6.3.

6.1. Accuracy on Static Data

6.1.1 2D Histograms

To demonstrate the ability of ISOMER to handle both numerical and categorical attributes, we used ISOMER to build a two-dimensional histogram on the attributes `model` (categorical) and `year` (numerical) of the `Car` table. Both test and training queries were of the form

```
SELECT * FROM Car WHERE model = x
AND year BETWEEN y1 AND y2
```

where x, y_1, and y_2 are variables. The queries were generated according to the data distribution. First, x was chosen randomly from all possible models; the probability for choosing a given model was proportional to the model's frequency in the database. Then y_1 and y_2 ($y_1 \leq y_2$) were chosen randomly from between the minimum and maximum years that occur with model x. Note that the execution of such queries gave us QFRs not only for two-dimensional predicates, but also for one-dimensional predicates, since predicates are applied one by one. Thus, we obtained QFRs such as $N(\texttt{model} = x)$ and $N(\texttt{model} = x,\ y_1 \leq \texttt{year} \leq y_2)$.

Figure 7 plots the error of the various approaches as a function of the number of training queries. Each histogram was allowed to have a maximum of $k = 175$ buckets. It can be seen that ISOMER significantly outperforms STGrid by providing much more accurate cardinality estimates. Also note that the error of ISOMER consistently decreases and then stabilizes, but never increases with the addition of more QFRs. Although STGrid also seems to possess this desirable property in case of static data, we show in the sequel that, for dynamic data, the error of STGrid may actually *increase* with addition of more QFRs. As expected, the optimizer performs poorly, mainly because of the limited multidimensional statistics it supports. That is, the optimizer keeps only a count of the number of distinct values for a group of columns, which is not useful for predicting the cardinality of range predicates on the numerical attribute `year`. Thus, the optimizer estimates are simply based on the independence assumption.

6.1.2 3D Histograms

In this experiment, we built a three-dimensional histogram on the highly correlated attributes `make`, `model`, and `color`. Both the training and test queries were of the form

```
SELECT * FROM Car WHERE make = x AND
model = y AND color = z
```

where x, y, and z are variables. The query-generation process was similar to that described in Section 6.1.1. First, a make x was chosen from the various makes according to the data distribution. Then from the various models corresponding to make x, a particular model y was chosen, again according to the data distribution. Finally a color z was chosen from the various colors corresponding to the (make, model) combination (x,y). ISOMER obtained QFRs for predicates of dimension 1, 2, and 3.

Figure 8 plots the error of the various approaches against the number of training queries. Each histogram was allowed to have a maximum of $k = 250$ buckets. ISOMER again outperforms STGrid by an even greater margin than in the 2D case, for two reasons. First, there are no partially overlapping buckets, because all attributes are categorical and hence the only predicates are equality predicates. Thus no bucket splits take place in ISOMER, so that a larger number of QFRs can be added to the histogram before the space budget is reached, thereby boosting accuracy. Second, STGrid performs poorly because it merely adjusts the

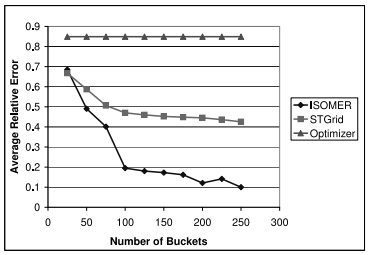

Figure 9. Error vs allowed number of buckets

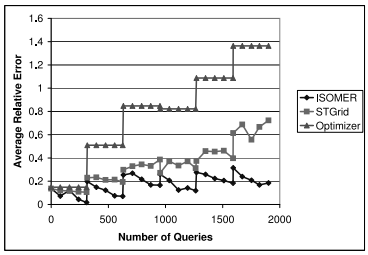

Figure 10. Updating: uniform to correlated

tuple counts in the buckets based on QFRs and does not use the QFRs to restructure the buckets. This lack of restructuring is a problem because buckets are determined from the initial one-dimensional statistics, that ignore the correlation between attributes. Since `make`, `model`, and `color` are much more strongly correlated than `model` and `year`, the performance of STGrid for this 3D histogram is worse than for the 2D histogram in Section 6.1.1.

6.1.3 Effect of Space Budget

To study the effect of the space budget on histogram accuracy, we used a 2D histogram on `model` and `year` as in Section 6.1.1.

Figure 9 plots the error of the ISOMER and STGrid histograms against the maximum number of buckets allowed, along with the optimizer error (the optimizer presently maintains a hardcoded, fixed number of buckets). The error was measured after 400 training queries had been executed and the corresponding QFRs used to refine the histogram. As expected, the error of both ISOMER as well as STGrid decreases as more buckets are allowed to be stored. However, ISOMER improves much more rapidly than STGrid as the space budget increases and outperforms STGrid at every value of the space budget.

6.2. Accuracy on Changing Data

To study the ability of ISOMER to deal with changing data, we worked with a 2D histogram on `model` and `year` as in Section 6.1.1. We interspersed the execution of training queries with the issuing of updates to the database. To specify the type of updates issued, we first define the notion of a *correlated tuple* and a *uniform tuple*. A correlated tuple is a tuple drawn from the real data distribution; it has all the correlations that the real database has. For example, since `make` and `model` are correlated, a correlated tuple can have the `(make, model)` value `(Honda, Civic)` but never `(Toyota, Civic)`. In contrast, a uniform tuple is generated by choosing the value of each attribute randomly and independently from the attribute's domain. For example, unlike a correlated tuple, the value of a uniform tuple can be any `(make, model)` combination.

We report the results of an experiment in which we changed the data distribution from uniform to correlated.[5] Specifically, we started out with a database consisting completely of uniform tuples. The optimizer gathered statistics over this database; these initial statistics remained fixed throughout the experiment. The training queries were executed and the gathered QFRs used to refine the histogram over time. After every 300 training queries, 20% of the tuples were updated from uniform to correlated tuples. This process was repeated 5 times, after which the entire database consisted only of correlated tuples.

Figure 10 plots the error of the various approaches against the number of queries executed. The spikes in the error curve that occur after roughly every 300 queries correspond to the times at which the database is updated. All the histograms start off very accurately because the data is uniform with independent attributes. For all approaches, the error increases when the data is updated. For ISOMER, however, the histogram evolves as queries are issued against the new data distribution, and the error decreases again. The STGrid error also decreases to some extent as the histogram is refined after updates. However, the improvement decreases as data becomes correlated. In fact, when more than 80% of the tuples have been updated, the addition of new QFRs tends to *increase* the overall error in STGrid. This is mainly due to the heuristic nature of STGrid, that does not preserve consistency when using new QFRs to refine the histogram. As expected, the optimizer error increases at each database update and never decreases, since the optimizer statistics are never refined.

ISOMER is not able to attain its original accuracy after updates have occurred. However, this phenomenon is expected because ISOMER imposes the uniformity assumption on any regions of the data distribution about which it lacks information. Hence ISOMER is bound to be more accurate on uniform than on correlated data.

6.3. Running Time

The results in Sections 6.1 and 6.2 show that in contrast to STGrid, ISOMER is robust and highly accurate across various data distributions and update scenarios. Nevertheless, the STGrid approach is very efficient because it uses a simple heuristic to refine the histogram. Although ISOMER

[5] We obtained similar results, not reported here, when changing the distribution from correlated to uniform.

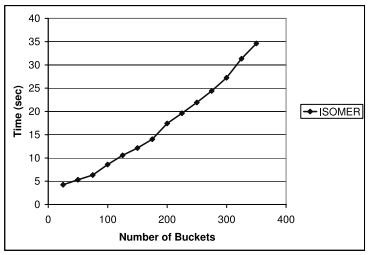

Figure 11. ISOMER running time vs number of buckets

is not as efficient as STGrid, we show in this section that the robustness of ISOMER comes at an affordable cost.

ISOMER's cost has two components. The first component is the cost of collecting QFRs, which has been shown [17] to be low (less than 5% overhead). Because this cost is incurred by any reactive approach, we do not consider it further. The second component of the ISOMER cost is the cost of refining the histogram using QFRs; this cost is incurred whenever ISOMER is invoked. To study this latter cost component, we work with a 2D histogram on `model` and `year` as in Section 6.2 and use changing data.

Figure 11 plots the total running time of ISOMER to add 4000 QFRs against the maximum number of buckets allowed in the histogram (A comparison with STGrid is not shown, since its running time is essentially 0). Note that although the worst case complexity is quadratic in the number of buckets (since all sibling pairs may be compared for bucket merging; Section 4.2), the actual running time is only slightly superlinear and remains well under a minute even for a moderately sized histogram with 350 buckets. In general, the number of buckets depends in a complicated manner on the number of distinct data values and the smoothness of the data distribution. We found that most of this time is spent in the process of reducing the histogram size to within the space budget, because the maximum-entropy solution has to be recomputed every time a QFR is discarded. We are exploring the use of the faster Newton-Raphson method for solving the maximum-entropy optimization problem in order to improve the overall running time of ISOMER.

7. Conclusions

We have described ISOMER, an algorithm for maintaining multidimensional histograms using query feedback. ISOMER uses the information-theoretic principle of maximum entropy to refine the histogram based on query feedback gathered over time. Unlike previous proposals for feedback-driven histogram maintenance, which lack either robustness (e.g., STGrid [1]), or efficiency (e.g., STHoles [4]), ISOMER is both reasonably efficient and robust to changes in the underlying data.

ISOMER can be extended in several ways to increase its utility in a database system. First, to handle a large number of attributes, ISOMER can be combined with techniques based on graphical models [7, 11]; such techniques divide the set of attributes into correlated subsets and maintain multidimensional statistics only for these subsets. Second, ISOMER can be extended to build histograms even on attributes in different tables, by using statistical views. Finally, ISOMER can be combined with a proactive approach in order to increase its robustness for queries that refer to data that has not been previously queried.

References

[1] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: building histograms without looking at data. In *SIGMOD 1999*.

[2] E. Amaldi and V. Kann. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoretical Computer Science*, pages 181–210, 1995.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[4] N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: A multidimensional workload-aware histogram. In *SIGMOD 2001*.

[5] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: How much is enough? In *SIGMOD 1998*.

[6] C. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. In *SIGMOD 1994*.

[7] A. Deshpande, M. Garofalakis, and R. Rastogi. Independence is good: dependency-based histogram synopses for high-dimensional data. In *SIGMOD 2001*.

[8] S. Guha, K. Shim, and J. Woo. REHIST: Relative Error Histogram Construction Algorithms. In *VLDB 2004*.

[9] D. Gunopulos, G. Kollios, V. Tsotras, and C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. In *SIGMOD 2000*.

[10] E. Jaynes. Information theory and statistical mechanics. *Physical Reviews*, 1957.

[11] L. Lim, M. Wang, and J. Vitter. SASH: A self-adaptive histogram set for dynamically changing workloads. In *VLDB 2003*.

[12] V. Markl et al. Consistently estimating the selectivity of conjuncts of predicates. In *VLDB 2005*.

[13] Y. Matias, J. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *SIGMOD 1998*.

[14] M. Muralikrishna and D. DeWitt. Equi-depth histograms for estimating selectivity factors for multidimensional queries. In *SIGMOD 1988*.

[15] V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB 1997*.

[16] J. Shore and R. Johnson. Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Trans. Information Theory*, 26(1):26–37, Jan. 1980.

[17] M. Stillger, G. Lohman, V. Markl, and M. Kandil. LEO - DB2's Learning Optimizer. In *VLDB 2001*.

[18] Computational Infrastructure for Operations Research . http://www.coin-or.org/.

[19] N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic multidimensional histograms. In *SIGMOD 2002*.