Combining Histograms and Parametric Curve Fitting for Feedback-Driven Query Result-Size Estimation

Arnd Christian König and Gerhard Weikum
Department of Computer Science,University of the Saarland
P.O. Box 151150, 66041 Saarbrücken, Germany
E-Mail: {koenig,weikum}@cs.uni-sb.de

Abstract

This paper aims to improve the accuracy of query result-size estimations in query optimizers by leveraging the dynamic feedback obtained from observations on the executed query workload. To this end, an approximate "synopsis" of data-value distributions is devised that combines histograms with parametric curve fitting, leading to a specific class of linear splines. The approach reconciles the benefits of histograms, simplicity and versatility, with those of parametric techniques especially the adaptivity to statistically biased and dynamically evolving query workloads.

The paper presents efficient algorithms for constructing the linear-spline synopsis for data-value distributions from a moving window of the most recent observations on (the most critical) query executions. The approach is worked out in full detail for capturing frequency as well as density distributions of data values, and it is shown how result size estimations are inferred for exact-match and range queries as well as projections and grouping. To a large extent, the developed methods can be generalized to multi-dimensional distributions, thus bearing the ability to capture correlations among attributes as well. Experimental studies underline the accuracy of the developed estimation methods, outperforming the best known classes of histograms.

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

1 Introduction

1.1 Deficiencies in the State of the Art

The effectivity of query optimization in database systems critically depends on the system's ability to estimate the costs of different query execution plans as accurately as possible. In most cases, the cost of a plan is dependent on the sizes of intermediate results produced by the query execution. Therefore, accurate estimation of these sizes is crucial to choosing a good plan and properly allocating the necessary resources, especially working memory.

A number of techniques for dealing with this problem have been studied in the literature, almost all of which can be categorized into three major classes: *histogram-based techniques* that capture statistical information about data-value distributions by means of counters for a specified number of data-value buckets, *parametric techniques* that approximate data-value distributions by fitting the parameters of a given type of function (e.g., polynomials of a given maximum degree), and query-specific *sampling* that take samples from the database to statistically estimate the intermediate result sizes for the query at hand (for survey material see [20, 1, 24, 2]). From a generalized perspective, all these approaches can be viewed as constructing an approximate representation, or *synopsis* [11, 9], of the data for the purpose of estimation (or even giving approximative query answers, which is not considered in this paper, however).

With modern OLAP tools and other forms of decision-support query generators, the query optimization takes place within the critical path of the query execution itself (i.e., there is no longer a distinction between the compile-time and run-time of a query). This rules out the relatively expensive sampling on a per query basis. Sampling is still very useful and, in fact, the method of choice for constructing histograms as a query-independent synopsis of the database's data-value distributions [3].

Histograms are traditionally a relatively static representation in that they do not easily adapt themselves to dynamically evolving value distributions (the only exceptional work being the methods for incremental

histograms proposed by [10], which, however, need to maintain a "backing sample" in addition to the histogram itself and are thus not exactly light-weight either). Furthermore, a histogram is a statistically unbiased representation in that it is not oriented towards a specific mix of queries and query-input parameters (e.g., the actual parameters used in filter conditions). In practice, however, it is often the case that the current workload bears a statistical bias in the sense that most queries, or the most important or performance-critical queries, refer to specific (ranges of) data values as input parameters. Therefore, it is desirable to express the same bias in the value-distribution representations and to maintain these in a form that can be dynamically adapted to evolving workload patterns with acceptable effort. For example, the representation should be more accurate for value ranges that are frequently referred to by current queries, and may tolerate inaccuracies for the other ranges.

With regard to the mentioned forms of adaptivity, parametric techniques appear to be an intriguing representation of data-value distributions, as they are based on fitting a parameterized curve with observations on data-values frequencies. Thus, these techniques can naturally accommodate biased observations from the actually executed queries, and can be adapted to dynamic shifts in the query patterns relatively easily by recomputing the curve fitting [4]. However, this does not mean that parametric techniques are an adequate representation in the first place. In fact, they seem to be suitable only for data-value distributions that resemble a closed-form distribution such as Zipf or Pareto distributions, whereas they would lead to poor estimations for highly irregular distributions, e.g., with multiple modes (i.e., non-adjacent, very frequent data values such as the top selling product numbers in a table of purchases). Unfortunately, such irregular distributions are typical for many real-life applications. It is only in conjunction with appropriate data-value permutations that many real-life distributions would resemble a closed-form distribution. Capturing the information on the necessary permutation is, however, out of the question from a system's viewpoint, as this would entail the same storage and lookup costs as a full-fledged index. Note in passing that for the same reason, various advanced forms of histograms, most notably, the family of V-optimal histograms that use attribute frequency as sort parameter [24], are not practically viable either.

1.2 Our Approach

So neither histograms nor parametric techniques are satisfactory in all regards; therefore, novel techniques are needed for adaptive result-size estimation in highly dynamic query-processing environments such as modern OLAP applications. In this paper we develop a new approach that combines histograms with parametric fitting into a specific class of *spline-based synopses* of data-value distributions. Such a synopsis is constructed in two steps:

First the active domain of an attribute (or multi-dimensional attribute combination) is divided into a number of buckets each of which represents a contiguous range of data values (multi-dimensional rectangle); a bucket is identified by its lower and upper value bounds (left lower and right upper corner). In contrast to the most simple type of so-called equi-width histograms, we allow the width of the value ranges to vary across buckets; these widths are one of the tunable degrees of freedom. In contrast to the advanced type of V-optimal histograms with attribute frequency as sort parameter, we restrict ourselves to buckets that represent contiguous values (rather than non-adjacent values with similar frequencies) to avoid having to build a full-fledged index for histogram lookups.

Second we represent the value distribution within a bucket as a spline function rather than a flat value. Thus we do not rely on a uniform distribution within a bucket but can capture trends within certain value ranges (e.g., increasing sales figures within a particular time period that corresponds to one bucket of the corresponding date attribute). For tractability, we will restrict ourselves to linear splines for each bucket. In contrast to the kind of representation that is referred to as "spline histograms" in [24], we do not require the splines to be continuous across all buckets. This way we can easily capture "jumps" in attribute value frequencies, for example. As our experimental results will show, this generalization is crucial for high accuracy. So in the one-dimensional case, we need to keep one additional value for each bucket (compared to histograms): in addition to the lower value bound and the frequency (i.e., average frequency in a histogram, and frequency of the lower value bound in our approach), we also store the slope of the linear spline function. The ratio in the memory demand between histograms and spline-based synopses will be taken into account in our experimental comparisons (Section 8).

For adaptivity, the basic idea is to record observations, or *feedbacks*, from a moving window of recent queries and construct a statistically biased representation from these recordings, possibly with focus on particularly important or critical queries (e.g., those that exhibit a large difference between the cost estimation by the optimizer and the actual execution cost). With n recorded observations of say the frequency of an attribute value and a continuous parametric representation with m buckets ($m < n$, and often even $m \ll n$) and k parameters to be fitted for each bucket, this leads to the so-called "knot placement problem" [6] which has been intensively studied in numerical mathematics and is known to have intractable complexity in full generality. Our restriction to linear splines and the relaxation to allow discontinuities at bucket bor-

ders render the problem tractable. We will present both optimal fitting algorithms and very efficient, sub-optimal heuristics that allow us to construct the entire synopsis on-the-fly as we obtain new observations from recent queries.

The adaptive representation sketched above is the core of our approach. The actual estimation procedures for the intermediate result sizes of simple selection queries are relatively straightforward. In addition, our representation is also suitable to capture not just the frequencies of data values but also the density of values within the data ranges of the buckets. This way we can represent the sparsity of values (e.g., price values in a fixed-point number domain from $ 0.99 to $ 10,000.00), which is important for estimating the result size of projections and especially grouping operations. Note that this aspect of selectivity estimation has been vastly neglected in the literature. Most prior work, including [24], makes rather simplistic assumptions about data-value densities within buckets, for example, assuming perfectly equi-distant data values. Further note that the data-value density has also a significant impact on the result sizes of range queries and equi-joins over sparsely populated domains. To this end, we will enhance the initial fitting of the bucket-specific values by an additional fitting step that takes into account feedback from range queries. So our approach of spline-based synopses that we pursue in this paper proves to be useful and, in fact, superior to prior methods, for a broad class of estimation problems.

The rest of the paper is organized as follows. Section 2 briefly reviews the state-of-the-art techniques for approximate data synopses and the estimation methods in query optimization. Section 3 introduces architectural assumptions and notation, setting the stage for Section 4, the paper's key section on the construction of spline-based synopses from exact-match query feedback. Section 5 shows how selectivity estimations for range queries are carried based on the spline-based synopses. Section 6 then introduces an additional fitting step to adjust the density values (i.e., number of distinct attribute values) that are kept in the synopses, taking into account additional feedback from range queries and leading to more accurate estimations for range queries. Section 7 shows presents the estimation procedures for projections and grouping queries. Finally, we present experimental results in Section 8 to demonstrate the practical benefits of the developed spline-based synopses. We conclude the paper with a brief discussion of various further extensions, including the treatment of multidimensional distributions and equi-join result sizes.

2 Related Work

Several techniques for dealing with the described problems have been proposed in literature, almost all of which can be categorized in three major classes: *Histogram-based techniques, parametric techniques* and *sampling* (for surveys, see [20, 1, 24]):

Histogram-based Techniques.
Histograms approximate value distributions by grouping attribute values into buckets, and estimating true attribute values and their frequencies based on assuming a uniform distribution within a bucket. Histograms are the most common form of data synopsis in practice and can be found in systems such as DB2, Informix, Ingres, Oracle, Microsoft SQL Server, Sybase and Teradata. The histogram accuracy depends on the type of histogram used: While *V-optimal(F,F)* histograms have been proven optimal for equi-joins and selections [15, 13, 16], they require a list of all attribute values in a bucket, which is impractical in real systems. A good compromise between accuracy and practicability is made by *MaxDiff(V,A)* histograms [24] or *V-optimal* using Sort Parameters other than Frequency [17] . Histograms can be efficiently constructed by sampling-based techniques [25, 10].

Another promising approach are Wavelet-based histograms [21], which so far have only been examined in the context of range queries, however.

Parametric Techniques. Parametric techniques (also known as curve-fitting or regression techniques) approximate value distributions using a mathematical distribution function with a limited number of free parameters. Values for these parameters are then chosen to fit the actual distribution. If the model is a good fit for the distribution, this provides an accurate and compact approximation; however, since the shape of the distribution is usually not known beforehand, this is often not the case.

To overcome this inflexibility, [28, 4] use a general polynomial function and apply least-squares fitting to choose its coefficients. [4] additionally uses query feedback; hereby, the approximation is able to adapt to changes in the value distribution as well as to locality in the actual query parameters, so as to be geared for more accurate estimations for frequently queried values or value ranges.

Sampling. These techniques compute their estimates by collecting and processing random samples of the data. Sampling techniques [12, 3, 7, 19] offer high accuracy and probabilistic guarantees on the quality of the estimation. However, since the sampling itself is typically carried out at the time of the approximation, the resulting overhead prohibits the use of sampling for query optimization. Therefore, in recent works [10, 8] techniques for incremental maintenance of random samples have been developed.

Since our main concern is a compact representation of data, and not its acquisition, we will now concentrate on the properties of parametric and histogram-based techniques, which both offer different advantages. While histograms offer accurate approximation for a wide range of data distributions, parametric techniques are quite limited in the distributions they approximate well. Even when polynomials [4] or rational functions [28] are used, approximation of distributions that exhibit large changes in the frequencies of adjacent attribute values and therefore are not well suited to approximation by means of a continuous function, lead to a degeneration of the approximation (see [4], for example, for the pathological effect of falsely estimating negative selectivities).

However, parametric techniques do have the important advantage of being *adaptive*, i.e. they can adapt to query locality, changes in the data distribution and – by assigning different *weights* to query feedbacks – to the importance of different estimations. While a certain adaptivity to changes in data distribution can be achieved in histograms, too, using techniques for incremental sampling [10, 8], the other forms of adaptivity, to our knowledge, have so far been restricted to parametric techniques.

3 Architectural Assumptions and Notation

3.1 Feedback-driven Architecture

Following the earlier proposals by [4] and, especially, [18] for adaptive selectivity estimation and dynamic re-optimization of query execution plans, we assume that the database system monitors the sizes (i.e., cardinalities in the sense of bags) of intermediate results of the executed queries. Like [18], we do not assume that we can observe all data values in an intermediate result, nor their frequency or density distribution. The rationale for this limitation is that such more insightful observations incur additional run-time overhead that could slow down the underlying database engine. For example, recording the size of a range selection is more or less a byproduct of the query execution with negligible overhead, whereas observing the density of the queried range (i.e., the number of distinct values) or the frequencies of the occurring values may require sorting or, at least, hash tables, and such additional resource consumption would often be out of the question.

Run-time observations, or *feedbacks*, are collected on-the-fly and captured in a suitable form of on-line statistics. In principle, each node in a query's operator tree can provide such feedback. However, we are mostly interested in leaf nodes, which are typically selections, or nodes close to the leaves, including joins that are performed early in the overall execution (e.g., the first join in a many-way join pipeline). The rationale for this consideration is that estimation errors for the early operators are the most troublesome in that they are the biggest factors in the error propagation for the entire execution plan [14]. In this paper, we will focus on feedback from exact-match selections (typically from index lookups, i.e., not necessarily table scans) and range selections. More general notions of feedback are the subject of future work.

Our approach separates the statistical feedback obtained from the executed queries from the approximative synopsis of value distributions that we maintain for selectivity estimations. This allows us to enforce certain filtering and aggregation steps in between the statistics collection and the actual estimations, according to the desired policies and resource allocations. For example, we may want to keep a relatively large number of observations from a moving window of the most recent queries for statistical confidence, but the synopsis should be much more compact. Then we could periodically reconstruct or incrementally maintain the synopsis from the observation window. In both cases the separation of observations and the synopsis serves to smooth out fluctuations.

As mentioned in the introduction, the pursued kind of statistics collection is inherently biased in that it is fed only by the actually executed queries. Sometimes, we may wish to incorporate an additional bias into an approximative synopsis, by focusing on expensive queries (e.g., those whose actually measured run-time costs exceed a certain limit), queries that are especially susceptible to large estimation errors (e.g., those where the difference between estimated and measured run-time costs exceed a certain limit), or simply those queries that are considered particularly important by a human expert (e.g., those which are expected to have a response time below a certain tolerated limit of say 10 seconds). All these forms of bias are readily taken into account by enforcing appropriate filter conditions when statistical observations are used for rebuilding or incrementally updating a synopsis.

3.2 Notation

We adopt the notation used in [24]. Without loss of generality, we consider only countable value domains (i.e., strings and fixed-point numbers, disregarding real numbers) and assume that these domains can be encoded as integers. Thus, we can define the *domain* $\mathcal{D} = \{0, 1, 2, \ldots, N-1\}$ of an attribute X as the set of all possible values of X, and the value set $\mathcal{V} \subset \mathcal{D}, \mathcal{V} = \{v_1, \ldots v_n\}$ is the set of values actually present in the underlying relation $\mathcal{R}$. In the context of this paper, $\mathcal{V}$ is usually observed through the feedback from the executed queries. Therefore, the value set $\mathcal{V}$ may actually be only a subset of the values in relation $\mathcal{R}$ if some value ranges are never touched by the feedback-relevant queries.

The *spread* s_i of v_i is defined as $s_i = v_{i+1} - v_i$

($s_0 = v_1$ and $s_n = 1$). The *density* of attribute X in a value range from a through b, $a, b \in \mathcal{D}$, is the number of unique values $v \in \mathcal{V}$ with $a \leq v < b$. The *frequency* f_i of v_i is the number of tuples in $\mathcal{R}$ of value v_i in Attribute X. The *cumulative frequency* c_i of v_i is the number of tuples $t \in \mathcal{R}$ with $t.X \leq v_i$. The *data distribution* of X is the set of pairs $\mathcal{T} = \{(v_1, f_1), (v_2, f_2), \ldots, (v_n, f_n)\}$. The *cumulative data distribution* of X is the set of pairs $\mathcal{T}^{\mathcal{C}} = \{(v_1, c_1), (v_2, c_2), \ldots, (v_n, c_n)\}$. The *extended cumulative data distribution* of X, denoted by $\mathcal{T}^{\mathcal{C}^+}$ is $\mathcal{T}^{\mathcal{C}}$ extended over the entire domain $\mathcal{D}$ by assigning zero frequency to every value in $\mathcal{D} - \mathcal{V}$. For the sake of a simple, uniform notation, we extend $\mathcal{T}$ by an artificial final tuple $(v_{n+1}, 0)$ with $v_{n+1} > v_n$.

4 Linear Splines as an Approximative Synopsis for Value Frequencies

To approximate a given value-frequency distribution $\mathcal{T}$, we partition the (observed) value set $\mathcal{V}$ into m disjoint intervals, henceforth called *buckets*, $b_i = [v_{low_i}, v_{high_i})$ in the following manner, where low_i and $high_i$ denote the *subscripts* of the values from $\mathcal{V}$ (i.e., not the actual values) that form the left and right bounds of the (left-side closed and right-side open) value interval covered by the bucket:

$$\forall i \in \{1, 2, \ldots, m-1\} : high_i = low_{i+1}$$
$$low_1 = 1,\ high_m = n + 1. \quad (1)$$

Unlike histograms, we approximate the frequency in an interval by a linear function, resulting in a linear spline function [5, 6] over the m buckets. Because our interest is in compact representation of distributions and not in the features more advanced forms of splines offer (smoothness, differentiability), we have chosen simple linear spline functions for this task. In contrast to previous approaches to spline-based histograms [25] we do not approximate $\mathcal{T}^{\mathcal{C}^+}$; therefore, we are not faced with the problem of forming approximation that is continuous over all intervals, thereby simplifying the problem of finding the optimal partitioning (see Subsection 4.2).

Using a linear function $f(x) = a_1 \cdot x + a_0$ to approximate the frequencies for the values x in a bucket leads to an improvement in accuracy, depending on the *linear correlation* [26] of the data within a bucket. First, we define $\bar{v}_{[low,high)} := \frac{1}{high-low} \sum_{l=low}^{high-1} v_l$ as the *average attribute value* within $[v_{low}, v_{high})$; analogously, we define the *average frequency* $\bar{f}_{[low,high)} := \frac{1}{high-low} \sum_{l=low}^{high-1} f_l$. The *linear correlation* for interval b_i is then defined as

$$r_{[low_i,high_i)} := \frac{\sum_{l=low_i}^{high_i-1} (v_l - \bar{v}_{[low_i,high_i)})(f_l - \bar{f}_{[low_i,high_i)})}{\sqrt{\sum_{l=low_i}^{high_i-1} (v_l - \bar{v}_{[low_i,high_i)})^2} \sqrt{\sum_{l=low_i}^{high_i-1} (f_l - \bar{f}_{[low_i,high_i)})^2}} \quad (2)$$

For each interval b_i, $r_{[low_i,high_i)} \in [-1, 1]$. In traditional histograms, the frequency in a bucket b_i is approximated by $avg\_f_{[low_i,high_i)}$. Using the least-squares fit as an error metric, this results in the overall error

$$err_{[low_i,high_i)} = \sum_{l=low_i}^{high_i-1} (f_l - \bar{f}_{[low_i,high_i)})^2.$$

In a spline-based synopsis, this error becomes :

$$spline\_err_{[low_i,high_i)} = (1 - r_{[low_i,high_i)}{}^2) \cdot err_{[low_i,high_i)}. \quad (3)$$

Obviously, $spline\_err_{[low_i,high_i)}$ is always less than or equal to $err_{[low_i,high_i)}$. This increase in accuracy is paid for by the fact that each bucket in the spline-based synopsis needs to store one more value than a bucket in a traditional histogram. This trade-off will be further examined by means of experiments in Section 8. Summing the error over all buckets in the synopsis, the overall error becomes:

$$ov\_spline\_err = \sum_{i=1}^{m} \left((1 - r_{[low_i,high_i)}{}^2) \cdot err_{[low_i,high_i)} \right). \quad (4)$$

In the following subsections we will develop algorithms for constructing a spline-based synopsis with m buckets for the n observed data-value frequencies in $\mathcal{T}$, aiming to minimize the overall error according to formula 4. This goal involves identifying the $m-1$ most suitable boundaries between buckets and the fitting for the linear approximation within each bucket. We will present a polynomial but not necessarily highly efficient algorithm for an optimal solution as well as much faster heuristic and thus suboptimal algorithms. All algorithms use the least-squares fitting with a bucket as a basic building block, which is presented in the following Subsection 4.1. The optimal and suboptimal algorithms for the global partitioning are then described in Subsections 4.2 and 4.3.

4.1 Fitting the Frequency Function within a Bucket

For the derivation of this basic building block suppose that the boundaries of a bucket are already fixed. For each bucket $b_i = [v_{low_i}, v_{high_i})$ we need to calculate the linear approximation $frq_i(x) = a_1 \cdot x + a_0$ of the attribute frequency that minimizes the squared error

$$spline\_err_{[low_i,high_i)} = \sum_{l=low_i}^{high_i-1} (frq_i(v_l) - f_l)^2. \quad (5)$$

This definition of the error over a bucket is equivalent to the definition 3 [26]; however, to evaluate formula 3, the coefficients a_1 and a_0, which are the unknowns of the fitting, do not have to be known. We will use this property in Section 4.2.

Using definition 5, finding frq_i becomes a problem of *linear least squares* fitting [26]; i.e. we are fitting the data $(v_l, f_l)_{l=low_i,\ldots,high_i-1}$ via the linear function $frq(x) = \sum_{l=0}^{1} a_l \cdot X_l(x)$ with $X_1(x) = x, X_0(x) = 1$ and the optimal a_1, a_0 to be determined. To do this, we construct a *design matrix* A of $(high_i - low_i) \times 2$ components, which are defined by $A_{l,h} = X_{l-1}(v_h)$, for $l = 1, 2$ and $h = low_i, \ldots, high_i - 1$. Furthermore, we define vector $b = (f_{low_i}, \ldots, f_{high_i-1})^t$. Now the fitting problem can be defined as finding $a = \binom{a_1}{a_0}$, which minimizes

$$spline\_err_i = |A \cdot a - b|^2.$$

This problem can now be solved by Singular Value Decomposition (SVD) [26, 1] in the following way: Let the SVD of A be

$$A = U \times \Lambda \times V^t \quad \text{, with } \Lambda = \begin{bmatrix} s_1 & & \\ & \ddots & \\ & & s_{high_i - low_i} \end{bmatrix}$$

with $s_1, \ldots, s_{high_i-low_i}$ being the singular values of A (i.e., the Eigenvalues of $A \times A^T$). Then a can be expressed as

$$a = \sum_{l=1}^{2} \left(\frac{U_{(i)} \cdot b}{s_i} \right) V_{(i)}$$

, $V_{(i)}$ and $U_{(i)}$ representing the i-th column of V, U. While computing the SVD of the design matrix causes considerable overhead, it is only computed m times, namely, once for each bucket of the final partitioning. Because of formula 3, we are capable of computing the optimal partitioning of $\mathcal{D}$ without calculating the exact frq_i. In addition to frq_i, each bucket b_i stores the number of attribute values contained in b_i. We will refer to this value as the *density* $D_i := high_i - low_i$ of bucket b_i.

4.2 Optimal Partitioning of $\mathcal{V}$

We are now interested in a partitioning such that the overall error (formula 4) is minimized. When arbitrary partitionings and continuous splines of arbitrary degree are considered, this is known as the *optimal knot placement problem* [5], which – due to its complexity – is generally solved only approximatively by heuristic search algorithms (for a detailed discussion, see [6]).

In our case, however, only linear splines are used and only members of $\mathcal{V}$ are candidates for bucket boundaries. Since the value for each $high_i$ is either low_{i+1} or v_{n+1} (see definition 1), we only need to determine the optimal lower bucket boundaries to compute:

$$opt\_err := \min_{\substack{(low_2,\ldots,low_m)\in\mathcal{V}^{m-1} \\ low_1 \le low2 \le \ldots \le low_m}} \sum_{l=1}^{m} (1 - r_{[low_l, high_l)}{}^2) \cdot err_{[low_l, high_l)} \quad (6)$$

Because the resulting spline function is allowed to be discontinuous over the chosen intervals $b_1, \ldots, b_m$, fitting the data in a bucket can be addressed separately for each bucket b_i. The main improvement in efficiency does, however, result from the fact that the following *principle of optimality* (also known as the Bellman principle) holds for our partitioning problem:

Theorem:

If for $l \ge 2$: $(low_l, low_{l+1}, \ldots, low_m) \in \mathcal{V}^{m-l+1}$ is an optimal partitioning of $[v_{low_{l-1}}, v_{high_m})$ using $m-l+2$ buckets, then $(low_{l+1}, low_{l+2}, \ldots, low_m) \in \mathcal{V}^{m-l}$ is an optimal partitioning of $[v_{low_l}, v_{high_m})$ using $m-l+1$ buckets.

Proof:

Because $(low_l, low_{l+1}, \ldots, low_m)$ is optimal, it minimizes

$$E := \sum_{i=l-1}^{m} spline\_err_{[low_i, high_i)}$$
$$= spline\_err_{[low_{l-1}, high_{l-1})} + \sum_{i=l}^{m} spline\_err_{[low_i, high_i)}.$$

Now assume that $(low_{l+1}, \ldots, low_m)$ is not optimal, i.e. there exists a partitioning $(low'_{l+1}, \ldots, low'_m)$ with $\sum_{i=l}^{m} spline\_err_{[low'_i, high'_i)} < \sum_{i=l}^{m} spline\_err_{[low_i, high_i)}$. But then $(low_l, \ldots, low_m)$ is not optimal either, for the partitioning $(low_l, low'_{l+1}, low'_{l+2}, \ldots, low_m)$ results in the overall error $E' = spline\_err_{[low_{l-1}, high_{l-1})} + \sum_{i=l}^{m} spline\_err_{[low'_i, hig'h_i)} < E$.

Because of this property, the problem of finding an optimal partitioning can be seen as a *dynamic programming problem* [27]. This allows us to formulate a recursive redefinition of formula 6: Define

$opt\_err_{low,\bar{m}}$:= the optimal overall error for fitting $[v_{low}, v_n)$ by $\bar{m}$ buckets.

$err_{[low,high)}$:= the approximation error $spline\_err_i$ for bucket $b_i = [v_{low}, v_{high})$.

Trivially, $opt\_err_{i,1} = err_{[i,n)}$. Then the overall error produced by the optimal partitioning is

$$opt\_err_{1,m} = \min_{l \in \{1,2,\ldots,n\}} err_{[1,l]} + opt\_err_{l,m-1}. \quad (7)$$

By keeping track of the partitioning, this equation can be used to compute an optimal partitioning in $O(m \cdot n^2)$ time, using $O(n^2)$ space. In the experimental section 8, we will refer to this algorithm as *OPTIMAL*.

4.3 Greedy Partitioning

Even if a spline-based synopsis were to be recomputed only occasionally, the cost for computing an optimal partitioning could be unacceptable when n is large. Therefore, we have also developed a greedy method of partitioning of $\mathcal{V}$, which results in a partitioning that is close to optimal while being much more efficient. The key idea is to start out with a large number (e.g., n) of trivial buckets, (e.g., each interval between two adjacent observed data values leads to one bucket), and

then gradually merge appropriately chosen adjacent buckets until we arrive at the desired number of m buckets. We will refer to this algorithm as *GREEDY-MERGE*.

In assessing the approximation quality of the buckets in each stage of the algorithm, we exploit the fact that, when merging the data distribution of 2 adjacent buckets $b_1 = [v_a, v_b)$ and $b_2 = [v_b, v_c)$ into one bucket $b' = [v_a, v_c)$, we can compute the resulting $spline\_err_{[a,c)}$ (formula 3) in constant time, if we maintain certain statistics for each bucket $b_i = [v_{low_i}, v_{high_i})$. These statistics are:

$$ff_{[low_i,high_i)} := \sum_{l=low_i}^{high_i-1} f_l^2, \qquad vv_{[low_i,high_i)} := \sum_{l=low_i}^{high_i-1} v_l^2,$$
$$vf_{[low_i,high_i)} := \sum_{l=low_i}^{high_i-1} v_l \cdot f_l, \qquad f_{[low_i,high_i)} := \sum_{l=low_i}^{high_i-1} f_l \quad (8)$$
$$v_{[low_i,high_i)} := \sum_{l=low_i}^{high_i-1} v_l, \ \bar{v}_{[low_i,high_i)} \text{ and } \bar{f}_{[low_i,high_i)}.$$

Now, when merging buckets $b_1 = [v_a, v_b)$ and $b_2 = [v_b, v_c)$ we can recompute these values for the resulting bucket in constant time:

$$\bar{v}_{[a,c)} = \frac{\bar{v}_{[a,b)} \cdot (b-a) + \bar{v}_{[b,c)} \cdot (c-b)}{(c-a)},$$

$\bar{f}_{[a,c)}$ is computed analogously, and the summations are obtained by simply adding the values for each bucket. These values now allow us to compute a bucket's linear correlation $r_{[a,c)}$ in constant time:

$$r_{[a,c)} = \frac{vf_{[a,c)} - f_{[a,c)}\bar{v}_{[a,c)} - v_{[a,c)}\bar{f}_{[a,c)} + (c-a)\bar{v}_{[a,c)}\bar{f}_{[a,c)}}{\sqrt{vv_{[a,c)} - 2\bar{v}_{[a,c)}v_{[a,c)} + (c-a)\bar{v}_{[a,c)}^2}\sqrt{ff_{[a,c)} - 2\bar{f}_{[a,c)}f_{[a,c)} + (c-a)\bar{f}_{[a,c)}^2}}$$

$$= \frac{\sum_{l=a}^{c-1} v_l f_l - \bar{v}_{[a,c)} \sum_{l=a}^{c-1} f_l - \bar{f}_{[a,c)} \sum_{l=a}^{c-1} v_l + \sum_{l=a}^{c-1} \bar{f}_{[a,c)}\bar{v}_{[a,c)}}{\sqrt{\sum_{l=a}^{c-1} {v_l}^2 - 2\bar{v}_{[a,c)} \sum_{l=a}^{c-1} v_l + \sum_{l=a}^{c-1} \bar{v}_{[a,c)}^2}\sqrt{\sum_{l=a}^{c-1} {f_l}^2 - 2\bar{f}_{[a,c)} \sum_{l=a}^{c-1} f_l + \sum_{l=a}^{c-1} \bar{f}_{[a,c)}^2}}$$

$$= \frac{\sum_{l=a}^{c-1} (v_l - \bar{v}_{[a,c)})(f_l - \bar{f}_{[a,c)})}{\sqrt{\sum_{l=a}^{c-1} (v_l - \bar{v}_{[a,c)})^2}\sqrt{\sum_{l=a}^{c-1} (f_l - \bar{f}_{[a,c)})^2}}, \text{ corresponding to } (2).$$

Now, the resulting error can easily be computed as

$$spline\_err_{[a,c)} = (1 - {r_{[a,c)}}^2) \cdot (ff_{[a,c)} - 2 \cdot \bar{f}_{[a,c)} \cdot f_{[a,c)} + (c-a) \cdot \bar{f}_{[a,c)}^2).$$

Based on these equations, we can now compute a nearly optimal partitioning using a greedy heuristics in $O(n \log_2 n)$ time. The algorithm partitions $\mathcal{V}$ into $\frac{n}{2}$ trivial buckets and merges the ones that lead to the smallest increase in the overall error, until only m buckets are left (see algorithm 1). The algorithm consists of 2 loops; the **for**-loop has $\frac{n}{2}$ iterations in which the error of merging the trivial buckets is computed, which can be done in constant time. The **repeat**-loop is executed $\frac{n}{2} - m$ times (each repetition reduces the number of buckets by one, there are $\frac{n}{2}$ buckets initially, and m upon termination), and executes 4 different types of operations:

1: Partition $\mathcal{V}$ into $\frac{n}{2}$ buckets $b_i = [v_{2i-1}, v_{2i+1})$.
2: **for** $l = 0$ to $\frac{n}{2}$ **do**
3: Compute the $error_{[2l-1,2l+1),[2l+1,2l+3)}$ resulting from merging the buckets b_l and $b_l + 1$ and insert the value into *queue* Q.
4: **end for**
5: **repeat**
6: Remove the minimal $error_{[a,b),[b,c)}$ from Q.
7: Merge the buckets $[a,b)$ and $[b,c)$.
8: Remove $error_{[a',a),[a,b)}$, $error_{[b,c),[c,c')}$ from Q.
9: Calculate the error of joining the new bucket with it's left and right neighbor (if exist); insert the resulting $error_{[a',a),[a,c)}$, $error_{[a,c),[c,c')}$ into Q.
10: **until** There are only m buckets left.

Algorithm 1: GREEDY-MERGE

1. Removing an item from the *priority queue* Q. We use an implementation of priority queues based on *Fibonacci heaps* [22], allowing the removal of an item in a queue of size n in $O(\log_2 n)$ time.
2. Merging two buckets, requiring constant time.
3. Calculating the error resulting from a merge, requiring constant time.
4. Inserting an item into Q, again requiring $O(\log_2 n)$ time.

Since each operation is carried out no more than three times, computing a greedy partitioning is of complexity $O(n \log_2 n)$.

The initial memory requirement is the space necessary for storing $\frac{n}{2}$ buckets, each of which contains 8 values (the lowest value stored in the bucket and the statistics detailed in definition 8), resulting in storage overhead of $4n$ values. If – due to the size of n – this overhead were not acceptable, one could further reduce the constant by choosing bigger initial buckets; for large n, this would probably not have any significant effect on the approximation quality of the partitioning. Choosing initial buckets of size 2^h leads to a storage overhead of $2^{3-h} \cdot n$ values.

Using a similar approach we developed an additional greedy partitioning-algorithm, which takes the opposite approach: Initially, all tuples are grouped in one bucket. Now, we will compute the split that leads to the greatest reduction in the overall error (formula 4) and execute it, resulting in an additional bucket. This is repeated, until (after $m - 1$ splits)

m buckets remain. Due to space constraints we do not go into more detail here. The algorithm requires $O(m \cdot n \log_2 n)$ time and $O(n)$ space. We will refer to this algorithm as *GREEDY-SPLIT*.

4.4 Running Times

In order to obtain an idea of the algorithms' efficiency in practice, we measured the running times of each partitioning method for different sizes of m and n (the v_i and f_i were choosen randomly, using a uniform distribution). The resulting running times for the partitioning methods *OPTIMAL* (*OPT*), *GREEDY-SPLIT* (*G-S*) and *GREEDY-MERGE* (*G-M*) for execution on a single processor of a *SUN UltraSPARC 4000/5000* (168 MHz) are shown in Table 1.

$n =$	500		1000		4000	
$m =$	10	50	10	50	10	50
OPT	0.59	2.13	2.78	11.04	46.52	191.6
G-S	0.009	0.008	0.019	0.021	0.069	0.068
G-M	0.001	0.007	0.004	0.020	0.042	0.153

Table 1: Running times in seconds

5 Result-Size Estimation for Range Queries

In this section we demonstrate how the spline-based synopsis of data-value distributions can be exploited by the procedure for estimating the intermediate result sizes of queries. Here we restrict ourselves to the estimation problem for simple range queries on a single attribute.

To estimate the result sizes of range queries, it is necessary to estimate the number of tuples whose attribute values for the range condition fall within an interval $[v_a, v_b)$, with $a, b \in \mathcal{D}$ being the query-specific actual parameters. The number of result tuples for such a range query is denoted by $S_{[a,b)}$. In order to account for skewed frequency distributions within a bucket, we need to inspect the spline functions of all buckets that intersect with the query range. This leads to the following estimation: If the query range happens to correspond to the boundaries of a bucket b_i, i.e. $a = low_i$, $b = high_i$, then $S_{[a,b)}$ is estimated by computing the cumulative frequency over the bucket, i.e., the sum of the value frequencies of the bucket, under the assumption that the attribute values that occur in the data and fall into b_i are equidistant within the bucket's spread. The number of values in the bucket, i.e., the bucket's density, is denoted as D_i. This yields the following expression for S^i, which we write instead of S when limited to a single bucket b_i:

$$S^i_{[low_i, high_i)} := \sum_{l=0}^{D_i - 1} frq_i\Big(low_i + l \cdot \tfrac{high_i - 1 - low_i}{D_i}\Big) \quad (9)$$

$$= \Big(\beta_i + \tfrac{\alpha_i \cdot (low_i + (high_i - 1))}{2}\Big) \cdot D_i + \tfrac{\alpha_i \cdot (low_i - (high_i - 1))}{2}.$$

Here the coefficients of the linear spline for b_i are denoted as α_i and β_i rather than a_1 and a_0, respectively, to make the dependency on bucket b_i more explicit. It is important to note that, although we assume an *identical spread* of the attribute values in b_i just like in histograms, our approach is more powerful since it allows us to properly capture a skew in the cumulative tuple distribution *within* each bucket.

For intervals that do not correspond to bucket boundaries, the density D_i needs to be multiplied with the fraction of the bucket covered by the query. If the query interval is completely contained in bucket b_i, i.e. $a \geq low_i, b < high_i$, we define $p_{[a,b),i} := \frac{b-a}{high_i - low_i}$ and obtain

$$S^i_{[a,b)} := \sum_{l=0}^{\lfloor D_j \cdot p_{[a,b),i} \rfloor - 1} frq_j\Big(a + l \cdot \tfrac{high_j - 1 - low_j}{D_j \cdot p_{[a,b),j}}\Big). \quad (10)$$

This equation can be transformed into the form $S^i_{[a,b]} = \gamma \cdot D_i + \delta$ with constants γ and δ that can be efficiently calculated from a, b, low_i, $high_i$, α_i, β_i, and D_i, but does not involve any summations over D_i. This ensures that the calculation of a range-query selectivity estimation requires only constant time for each bucket. Finally, the result size of range queries for intervals spanning more than one bucket can easily be estimated by partitioning $[v_a, v_b)$ into $[v_a, v_{high_j})$,$[v_{low_{j+1}, high_{j+1}})$, $\ldots$,$[v_{low_{j+l-1}}, v_{high_{j+l-1}})$,$[v_{low_{j+l}}, v_b)$ and summing the estimated result sizes obtained by the above formulas:

$$S_{[a,b)} := S^j_{[a, high_j)} + \Big(\sum_{h=1}^{l-1} S^{j+h}_{[low_{j+h}, high_{j+h})}\Big) + S^{j+l}_{[low_{j+l}, b)}. \quad (11)$$

6 Improved Fitting of Value Densities

So far the density of a bucket, i.e., the number of distinct values that fall into the bucket's value interval, is simply based on counting the distinct exact-match queries in the observed statistics (see Section 4.1). Once we have formed buckets, we lose the information on the value distribution within a bucket and need to resort to the assumption of equidistant values. This assumption, however, can lead to significant distortions in the result-size estimations for range queries if there is a skew in the actually occurring values. To illustrate this kind of estimation error, consider a simple example with a bucket b_i ranging from $v_{low_i} = 10$ to $v_{high_i} = 20$ with actually observed value-frequency pairs $\{(v_l, f_l) \mid low_i \leq l \leq high_i - 1\} = \{(10, 100), (11, 90), (12, 80), (13, 70), (19, 10)\}$. So the majority of the observed values lie in the first half of the bucket, and these values are also much more frequent. The spline function that results from the fitting

according to Subsection 4.1 is $frq_i(v) = -10v + 200$ (which actually has zero error for the particularly assumed frequencies in this bucket). The density D_i of the bucket simply equals $high_i - low_i = 5$, i.e., the number of observed values. Now suppose that range queries mostly refer to intervals in the first half of the bucket, which may not be surprising at all given that this is where we have observed more data values. (But note that, in general, the distribution of the data does not have to be correlated with the distribution of the actual query parameters.) Then, not taking into account the skew in the value density within the bucket would lead to a significant estimation error for such range queries. For example, the result size of a range query $S_{[10,16)}$ for an interval from 10 through 16 would be estimated as $frq(10) + frq(13) + frq(16) = 100 + 70 + 40 = 210$ based on the equidistant-values assumption and the fact that the query range covers $\frac{3}{5}$ of the bucket interval. The actual result size, however, is $100 + 90 + 80 + 70 = 340$, quite a large deviation from the estimated size.

To rectify the discussed kind of problem, we adjust the density values D_i that are kept for the buckets of a spline-based synopsis by considering additional feedback from range queries. This leads to another fitting problem that aims to minimize the estimation error for range queries. Note, however, that this is addressed as a subsidiary issue which leaves the previously completed fitting of the frequency values (according to Section 4) invariant and only aims to adjust the density values for reducing the estimation error. Again, we are driven by adapting our estimations to the skew in queries, to obtain a more accurate estimation for frequently queried intervals, at the expense of a larger error for less important ones. For this task, we examine a set of query feedbacks from range queries $F := \{([a_j, b_j), size_j), j = 1, \ldots, k\}$ with $size_j = \sum_{\nu | a_j \leq v_\nu < b_j} f_\nu$ being the total number of tuples (i.e., counting duplicates) in interval $[a_j, b_j)$. We are interested in finding the optimal D_i, such that

$$\tilde{E} := \sum_{j=1}^{k} (size_j - S_{[a_j,b_j)})^2$$

is minimized where the $S_{[a_j,b_j)}$ are the estimated result sizes for the most recent k observed range queries. Now, $S_{[a_j,b_j]}$ can be rewritten as (cf. formula 9):

$$S_{[a_j,b_j)} = \Big(\sum_{l=0}^{m} \gamma_{j,l} \cdot D_i\Big) + \delta_j \qquad (12)$$

The problem of finding the optimal D_i values that minimize the error $\tilde{E}$ can be formulated as a problem of *linear least squares*. Here, the *design matrix* A holds $k \times m$ components. We define

$$A_{jl} := \gamma_{j,l} \text{ for } j = 1, \ldots, k \text{ and } l = 1, \ldots, m$$

and $b = \begin{pmatrix} size_1 - \delta_1 \\ \vdots \\ size_k - \delta_k \end{pmatrix}$. Now the fitting problem can be defined as finding $a = \begin{pmatrix} D_1 \\ \vdots \\ D_m \end{pmatrix}$ that minimizes $\tilde{E} := |A \cdot a - b|^2$.

We can again compute a using an SVD [26], analogously to Section 4.1. One drawback of this approach is that the space requirement of A and b increase in proportion to the number of query feedbacks k, resulting in considerable overhead when "fitting" long series of query feedbacks. However, since the feedback arrives incrementally, we can use an iterative fitting technique know as the *recursive least squares regression* [29]. For this incremental approach, we only need to maintain two $m \times m$ matrices, as opposed to a $k \times m$ matrix. These matrices are updated with each feedback (for a detailed description of *recursive least squares regression* in the context of database-query feedback, see [4]). Since m (i.e, the number of buckets) is a rather small constant, the resulting overhead for the fitting is affordable.

7 Result-Size Estimation for Projections and Grouping Queries

The density of attribute values, i.e., the number of distinct values within certain ranges, is the decisive information for estimating the result sizes of projections with duplicate elimination. Although such projections themselves are not among the most critical operators as far as query optimization is concerned, implicit projections take place also in grouping and aggregation operators. The latter are definitely among the most important operators in modern decision-support applications. In this section we show how the size of projections in combination with a range-filter condition can be estimated from our approximative synopses.

We denote the number of distinct values that fall into interval $[v_a, v_b)$ by $P_{[a,b)}$. When the query range corresponds to the boundaries of a bucket b_i, $P^i_{[low_i, high_i)}$ is simply approximated by D_i. If $[v_a, v_b)$ is completely contained in b_i, $P^i_{[a,b)} := p_{[a,b),i} \cdot D_i$, where $p_{[a,b),i}$ is the fraction of the bucket's interval covered by the query range (see Section 5). Finally, for intervals spanning multiple buckets, the result-size estimation can be obtained using an partitioning similar to the one employed in formula 11:

$$P_{[a,b)} := P^j_{[a,high_j)} + \Big(\sum_{h=1}^{l-1} P^{j+h}_{[low_{j+h}, high_{j+h})}\Big) + P^{j+l}_{low_{[j+l,b)}}.$$

8 Experimental Results

In this section we present experimental results, using synthetic data sets. In order to illustrate the effectiveness of our approach, we compare its accuracy for

several estimations problems to histograms proposed in [24, 25].

8.1 Experimental Comparison of the Estimation for Exact-Match Queries

We have experimentally studied the accuracy of several techniques for approximating and estimating the result size of exact-match queries of type $size_i = \{t \in R \mid t.A = v_i\}$. We compared the spline-based synopsis developed in Section 4 to the well-studied *equi-depth* and *equi-width* as well as *MaxDiff(V,A)* [25] and *V-optimal(V,F)* [17] histograms.

Because our technique stores four values per bucket (low value bound, frequency of the low value bound, slope of the frequency curve, and density), whereas traditional histograms store only three (low value bound, average frequency, density), our technique uses only $\frac{3}{4}$ of the buckets of traditional histograms in the experiments. As error measure we again use the squared-error norm: with $size_i$ denoting the actual result size of query i and S_i' the size-approximation, we measure $\sum_i (size_i - S_i')^2$. We have studied a variety

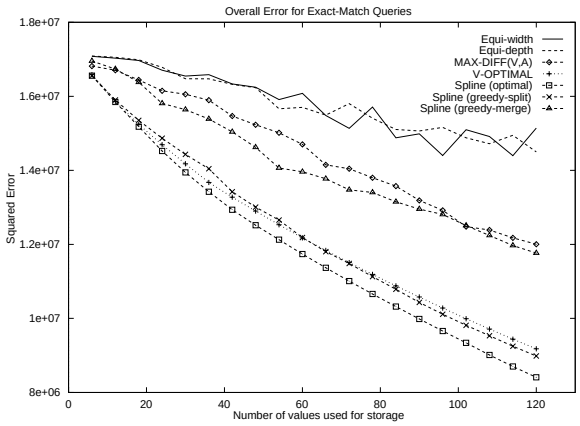

Figure 1: Squared error of various techniques

of data distributions, including the synthetic data sets from earlier studies on histograms, real-life data sets like the NBA database [23], and also completely random data. All of these produced comparable results as far as the estimation accuracy of the studied algorithms is concerned. Here we restrict ourselves to presenting the results for a single synthetic distribution: The value set size is $n = 500$, the domain size is $N = 4096$, and the relation size is $|R| = 10^5$. The data distribution was generated by assigning the individual frequencies randomly, using an uniform distribution. Note that this poses some form of stress test for the various approximation methods, as the data exhibits both highly skewed frequencies and high irregularity. Furthermore, the data distribution exhibits only low linear correlation, thereby minimizing the gain resulting from the use of linear spline approximation. For this experiment, Figure 1 shows the squared error of the various methods under comparison as a function of the number of values that each method can store with a given amount of memory space. So the number of buckets is 1/4 of the number of stored values for the spline-based synopsis, and 1/3 of the number of stored values for the histograms. As the figure shows, the optimal spline-based synopsis performs better than all histograms, while *GREEDY-SPLIT* and *V-optimal* histograms offer comparable accuracy. In data distributions of higher linear correlation, the optimal spline-based synopsis again outperforms all other methods, however, the *GREEDY-SPLIT* and *GREEDY-MERGE* partitioning also surpass the accuracy of all histogram methods. To illustrate the enhanced accuracy of our approach, Figure 2 shows two examples for the fitting of a completely random distribution (i.e., uniformly distributed frequencies) with $n = 30$ different attribute values. Both MaxDiff histograms and the spline-based synopsis adapt themselves to the data in that they choose smaller bucket widths for ranges with high fluctuations. In addition, however, the spline-based synopsis enables us to capture trends within certain ranges of the data, whereas histograms merely reflect average frequencies in these ranges.

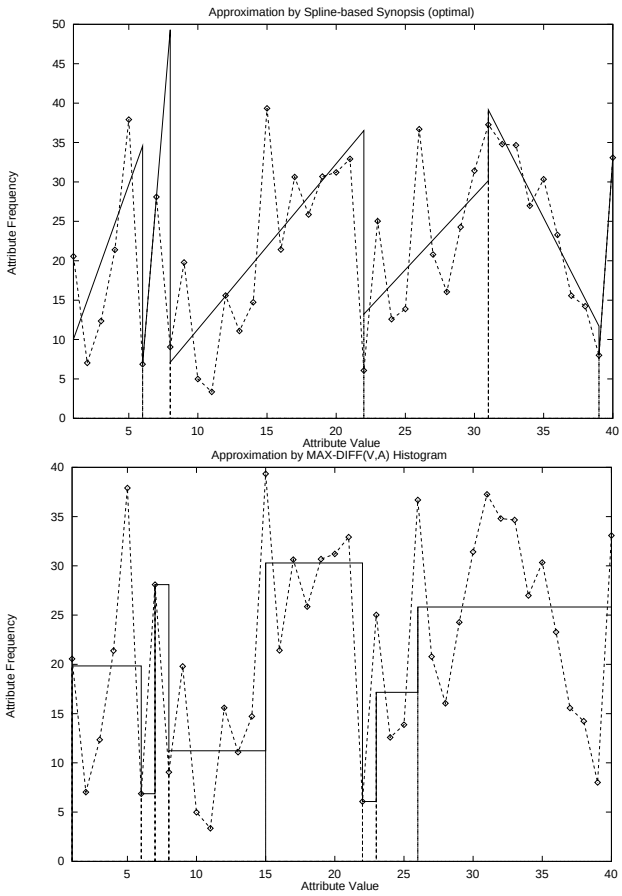

Figure 2: Illustration of approximation quality

8.2 Experimental Comparison of the Estimation for Range-Queries

Using the synthetic data set described in Subsection 8.1, we have also examined the accuracy of the various techniques for estimating the size of range queries, using the techniques of Section 4. We considered the following types:
$A: \{X < b \mid b \in \mathcal{D}\}$, $C: \{a \leq X < b \mid a, b \in \mathcal{D}, a \leq b\}$,
$B: \{X < b \mid b \in \mathcal{V}\}$, $D: \{a \leq X < b \mid a, b \in \mathcal{V}, a \leq b\}$,
$E: \{a \leq X < a+\Delta \mid a \in \mathcal{D}\}$, $F: \{a \leq X < a+\Delta \mid a \in \mathcal{V}\}$
with a, b randomly chosen from the underlying value sets according to a uniform distribution.

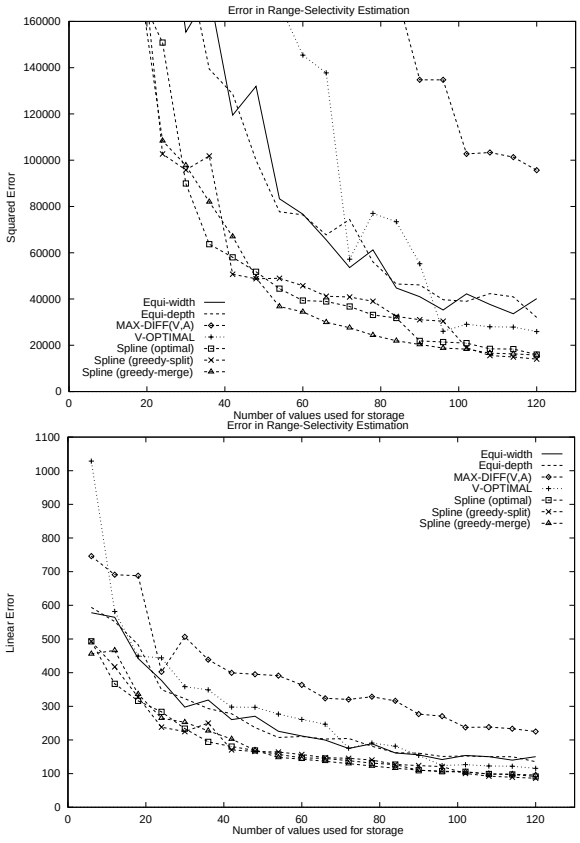

Figure 3: Squared and linear error for range queries

In Figure 3 we show the experimental results for query type A; the experiments for the other query types showed the same qualitative results and are thus omitted here. In order to illustrate that our algorithm – while geared for the least-squares error metric – also results in good estimations with respect to other error metrics, we show the average linear error of a query in addition to the squared error. The *MaxDiff(V,A)* histogram exhibits the worst accuracy in its estimations because of the intricacies of the data set: the data contains many large "jumps" in the frequencies of subsequent attribute values, thereby producing several very small buckets. Interestingly, the optimal partitioning does not always lead to the optimal overall error for range queries; this is due to the fact that the partitioning is initially geared towards minimizing the error for exact-match queries rather than range queries. However, the chart shows that all partitioning techniques for spline-based synopses perform almost equally well, surpassing the histogram-techniques. Again, when using data of higher linear correlation, this trend becomes more clear.

9 Conclusion and Future Work

In this paper we have developed a novel approach to query result-size estimation based on a feedback-driven approximative representation using linear splines that are allowed to be discontinuous across buckets. To the best of our knowledge, this is the first approach that combines the adaptivity with regard to evolving query locality (i.e., actual parameters of the queries) with a very high accuracy of the estimations even in the presence of highly irregular, skewed distributions. In fact, our experimental studies have shown that the spline-based synopses are superior to histograms in terms of accuracy.

The main issue that we have addressed and solved is the fitting of the free parameters of a spline-based synopsis to the observed query feedbacks. This has involved both finding appropriate bucket boundaries – the partitioning problem – and a least-squares regression for each bucket. We have seen that the initial feedback from exact-match queries that we have considered can be enhanced by additional feedback from range queries for an improved fitting of the density value kept in each bucket. However, it has to be noted that there is the unresolvable tension in fitting these values for different query types such as range queries versus projections or grouping/aggregation queries (unless we wanted to keep different synopses for different query types at the expense of more memory space). Our preference for range queries was mostly driven by the observation that estimation errors in the earliest operators typically have the most severe impact as the error propagates through the operator tree.

Extending our technique for spline-based synopses to multiple dimensions (i.e., attribute combinations) is relatively straightforward; however, some open issues remain. When fitting d-dimensional data $\big((v_{i,1}, \ldots, v_{i,d}), f_i\big)_{i=1,\ldots,n}$, we partition the data space into a mesh consisting of m^d d-dimensional intervals determined m observed data points. While the approximation of a multidimensional frequency function for each bucket is straightforward (fitting via SVD for $d+1$ coefficients), we are in the process of studying good partitioning techniques that can be computed in acceptable time.

The techniques described in this paper can be further extended to the fitting of feedback generated by equi-join queries. When computing the result size of a join over an interval $b_i = [a, b)$, the decisive factor for an accurate estimation of the join size is the accuracy in estimating the number of joining values t_i from the

two relations whose join-attribute values fall into b_i. While a number of heuristics that estimate t_i from the interval's density (i.e., its number of distinct values) have been put forth in the literature, to the best of our knowledge none of these techniques are very accurate for arbitrary distributions of the occurring join-attribute values. An approach that we are pursuing is to exploit feedback from equi-join queries by observing and keeping the number of join tuples within observable value intervals for each pair of frequently joined relations (while resorting to simpler heuristics for infrequent join relations). For a given join with t_i joining values in a join-attribute interval $b_i = [a, b)$, the join result size would be estimated from the value-frequency synopses of the two relations $\mathcal{R}_1, \mathcal{R}_2$ as follows: $S_{[a,b)} = \sum_{i=0}^{t_i-1} frq_i^{\mathcal{R}_1}\big(a + \frac{i}{t_i}(b-a)\big) \cdot frq_i^{\mathcal{R}_2}\big(a + \frac{i}{t_i}(b-a)\big)$.

Our approach would then aim to determine a fitting for the t_i values into a compact synopsis (whose number of buckets is much smaller than that of the observed t values and result sizes). Unfortunately, $S_{[a,b)}$ can not be rewritten as a function that is linear in t_i, thereby making least-squares fitting infeasible. Instead, $S_{[a,b)} = \Lambda \cdot t_i + \Phi + \frac{\Delta}{t_i}$, which can be fit with adequate accuracy using iterative numerical techniques [26].

References

[1] D. Barbará, W. DuMochel, C. Faloutsos, P.J. Haas, J.M. Hellerstein, Y. Ioannidis, H.V. Jagadish, T. Johnson, R. Ng, V. Poosala, K.A. Ross, and K.C. Sevcik. The New Jersey Data Reduction Report. Technical report, 1997.

[2] S. Chaudhuri. An overview of query optimization in relational systems. In *ACM PODS*, pages 34–43, 1998.

[3] S. Chaudhuri, R. Motwani, and V.R. Narasayya. Random sampling for histogram construction: How much is enough? In *Proc. of the ACM SIGMOD Conference*, pages 436–447, 1998.

[4] C.M. Chen and N. Roussoploulos. Adaptive selectivity estimation using query feedback. In *Proc. of the ACM SIGMOD Conference*, pages 161–172, May 1994.

[5] C. de Boor. *A practical guide to splines.* Springer-Verlag, 1978.

[6] P. Dierckx. *Curve and Surface Fitting with Splines.* Monographs on numerical Analysis. Oxford Science Publications, 1993.

[7] S. Ganguly, P.B. Gibbons, Y. Matias, and A. Silberschatz. Bifocal sampling for skew-resistant join-size estimation. In *Proc. of the ACM SIGMOD Conf*, 1996.

[8] P.B. Gibbons and Y. Matias. New Sampling-Based Summary Statistics for Improving Approximate Query Answers. In *Proceedings of the ACM SIGMOD Conference*, 1998.

[9] P.B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. Tech. report, Bell Labs, 1998.

[10] P.B. Gibbons, Y. Matias, and V. Poosala. Fast Incremental Maintenance of Approximate Histograms. In *Proc. of the 23rd VLDB Conference*, 1997.

[11] Phillip B. Gibbons, S. Acharya, Y. Bartal, Y. Matias, S. Muthukishnan, V. Poosala, S. Ramaswamy, and T. Suel. AQUA: System and techniques for approximate query answering. Tech. report, Bell Labs, 1998.

[12] Peter J. Haas. Selectivity and cost estimation for joins based on random sampling. *Journal of Computer and System Sciences*, pages 550–569, 1996.

[13] Y. Ioannidis. Univeratility of Serial Histograms. In *Proceedings of the 19th VLDB Conference*, pages 256–267, December 1993.

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

[15] Y. Ioannidis and S. Christodoulakis. Optimal Histograms for limiting Worst-Case Error Propagation in the Size of Join Results. In *ACM TODS*, 1993.

[16] Y. Ioannidis and V. Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. In *Proceedings of the ACM SIGMOD Conference*, pages 233–244, May 1995.

[17] H. V. Jagadish, N. Koudas, S. Mutukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal Histograms with Quality Guarantees. In *Proc. of the 24th Int. VLDB Conf.*, pages 275–286, August 1998.

[18] N. Kabra and D.J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *Proceedings of the ACM SIGMOD Conference*, 1998.

[19] Y. Ling and W. Sun. An evaluation of sampling-based size estimation methods for selections in database systems. In *Proc. of the ICDE Conf.*, 1995.

[20] M.V. Mannino, P. Chu, and T. Sager. Statistical Profile Estimation in Databbase Systems. In *ACM Computing Serveys*, 1988.

[21] Y. Matias, J.S. Vitter, and M. Wang. Wavelet-Based Histograms for Selectivity Estimation. In *Proc. of the ACM SIGMOD Conf.*, pages 448–459, 1998.

[22] K. Mehlhorn, S. Näher, and C. Uhrig. LEDA: Library of Efficient Data types and Algorithms. available via ftp:mpi-sb.mpg.de, 1997.

[23] NBA Statistics for the 91-92 Season. available under: ftp:olympos.cs.umd.edu.

[24] V. Poosala. *Histogram-based Estimation Techniques in Database Systems.* PhD thesis, University of Wisconsin-Madison, 1997.

[25] V. Poosala, Y.E. Ioannidis, P.J. Haas, and E.J. Shekita. Improved Histograms for Selectivity Estimation or Range Predicates. In *Proc. of the 1996 ACM SIGMOD Conference on the Management of Data*, pages 294–305. ACM Press, 1996.

[26] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P Flannery. *Numerical Receipes in C, The Art of Scientific Computing.* Cambridge University Press, 1996.

[27] M. Sniedovich. *Dynamic Programming.* Marcel Dekker, Inc., 1992.

[28] W. Sun, Y. Ling, N. Rishe, and Y. Deng. An instant and accurate size estimation method for joins and selections in an retrival-intensive environment. In *Proc. of the ACM SIGMOD Conference*, pages 79–88, 1993.

[29] P. Young. *Recursive Estimation and Time-Series Analysis.* Springer-Verlag, 1984.