Smooth Interpolating Histograms with Error Guarantees

Thomas Neumann[1] and Sebastian Michel[2]

[1] Max-Planck-Institut Informatik, Saarbrücken, Germany
neumann@mpi-inf.mpg.de
[2] École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
sebastian.michel@epfl.ch

Abstract. Accurate selectivity estimations are essential for query optimization decisions where they are typically derived from various kinds of histograms which condense value distributions into compact representations. The estimation accuracy of existing approaches typically varies across the domain, with some estimations being very accurate and some quite inaccurate. This is in particular unfortunate when performing a parametric search using these estimations, as the estimation artifacts can dominate the search results. We propose the usage of linear splines to construct histograms with known error guarantees across the whole continuous domain. These histograms are particularly well suited for using the estimates in parameter optimization. We show by a comprehensive performance evaluation using both synthetic and real world data that our approach clearly outperforms existing techniques.

1 Introduction

Query optimization is largely based on selectivity (and thus cardinality) estimations. Typically, these cardinalities are the most significant parameters of cost estimations. The selectivity estimates are derived from precomputed statistical information, usually in the form of histograms. A histogram contains a condensed representation of the value distribution of one attribute (or multiple attributes for multidimensional histograms). When a query contains a predicate on that attribute, the corresponding histogram can be used for selectivity estimation. For these estimations, the histograms are probed with values derived from the query itself. These are only a few constants for most of the queries, but in some situations the query optimizer itself will generate new constants.

As a showcase application for these kind of query optimization problems we briefly discuss a typical issue in the area of top-k query processing. Among the ample work on top-k query processing, the family of threshold algorithms (cf, e.g., [1–3]) stands out as an extremely efficient and highly versatile method. However, none of these algorithms can be directly applied in a widely distributed setting, as they still incur an unbounded number of message rounds. In contrast, state-of-the-art algorithms for distributed top-k aggregation use a fixed number of communication rounds to bound latency. The first algorithm in this family was the TPUT (Three-Phase Uniform Threshold) algorithm [4], in which the top-k query is translated into a range scan with appropriate postprocessing. Yu et al [5] present a modification of TPUT where the range bounds are adapted to the specifics of the value distributions. Along these lines stands our own work [6] on distributed top-k query optimization. The basic idea behind the algorithms presented in [4, 5, 7] is the transformation of a top-k query into the union of range queries where the range is determined by an initial retrieval phase. In a simple example of a top-k query that involves two index lists with a given range of 0.9, i.e., all documents with an aggregated score of at least 0.9 are potential candiates, for summation the second retrieval phase can be written as

(**select** id **from** list1 **where** score≥0.5)
union all
(**select** id **from** list2 **where** score≥0.4)

Although this query requires some post processing, as it produces a super set of the original top-k query, it can be executed more efficiently. As for normal range queries, histograms can be used to estimate the number of qualifying data items. Note however that the constants 0.5 and 0.4 in this query do not occur in the original top-k query and that they were chosen somewhat arbitrarily. In general any pair (a, b) of score thresholds would be suitable as long as $a+b \leq 0.9$. During query optimization, histograms are probed for many different (generated) values that will often not occur in the actual data.

This has consequences for the requirements on histograms. The two main requirements for constant optimization are:

1. smooth estimations, i.e., the estimations do not create artificial extrema. Formally, $\forall_{[a,b]\supseteq[a',b']} H([a, b]) \geq H([a', b'])$, where $H([., .])$ is the estimator for a range query.
2. interpolating estimations, i.e., the estimation works well for any value in the potential domain.

The smoothness requirement implies that numerical methods can be used on the histogram. This is sometimes violated by more complex histograms (e.g., [8]) when modeling the CDF. As this is to some extent application specific, we do not elaborate on this in the paper, although our histograms satisfy the requirement. As we will see in Section 5, the interpolation requirement is not satisfied by many existing approaches as they usually concentrate on creating histograms to accurately represent the actual data while rarely occurring or non-occurring attribute values cause large errors.

In this paper, we propose the usage of continuous linear splines to model data distributions and present algorithms to efficiently construct interpolating splines that have a low error guarantee across the whole domain, and that allow smooth selectivity estimations. These spline histograms have some further nice properties for top-k processing, in particular, they are reversible and allow for determining the range bounds when the cardinality is known. Moreover, the spline histograms are not limited to information retrieval settings, the experimental results show that they are superior to existing approaches even if the additional properties are not required. The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 reasons about the usage of splines for interpolation. Subsequently, Section 4 presents an optimal and a greedy algorithm to construct spline histograms. Section 5 presents the experimental evaluation.

2 Related Work

Histograms are a well studied field in database research. In fact, there are so many different kinds of histograms that we can only highlight closely related work here, and refer to [9] for a comprehensive overview. In our work, we concentrate on histograms that can be used for selectivity estimations of range queries. More general histograms exist, for example for multidimensional data [10], but this is beyond the scope of this work.

Spline based histograms have been mentioned in the literature before [11], but there the knot placement problem is considered as too difficult. The paper mentions that heuristic spline construction is possible, but the results shown in [11] indicate that these heuristic splines are not relevant (the paper explicitly classifies them as "poor histograms"). Another spline based histogram approach has been proposed in [8], that in contrast to our work, disregards spline features like smoothness and continuity. It partitions the value domain into buckets, computes the frequencies inside each bucket, and fits a linear function to the frequencies using least squares fitting. As we will see in Section 5, this bucket-wise fitting can cause huge errors near the bucket boundaries.

A very well known histogram construction technique are V-Optimal histograms [12]. The algorithm minimizes the frequency variance inside the buckets, which results in a good accuracy in general. It is not well suited for continuous data, where the frequencies are usually all equal to 1 (see Section 5). Recently histogram approaches have been suggested that deviate further from the classical view of histograms. Two examples are [13, 14], that construct a wavelet representation of the original data and then discards coefficients to reduce the space consumption. This is very similar to lossy compression techniques. One difficulty is that the data has to be discretized first, and that the accuracy is sensitive to the chosen approximation, as we will see in Section 5.

3 Spline Interpolation

The goal of histogram construction is to find a compressed representation of the raw data that still allows for accurate estimations. We propose the usage of splines; partially because they can represent distributions efficiently, and partially because we can construct them in way that provides smooth estimations with error guarantees. In this section we discuss the general approach of using spline histograms, the concrete algorithms are shown in Section 4.

3.1 Data Representation

The raw data itself consists of a multiset of values, potentially from a continuous domain. For query optimization, the histogram is typically queried in two different ways. Either it should determine how many occurrences of an attribute that have a certain value are in the data (*point query*) or how many occurrences of an attribute are within a given range (*range queries*). We concentrate on range queries here, as point queries on a continuous domain are not meaningful in general since the expected number of occurrences is zero. On a discrete domain, range queries can be used to approximate point queries.

The best representation for our purpose is the *cumulative distribution function* (CDF). For each point in the domain it gives the number of values less or equal to this point, which can easily be used to answer range queries. Additionally, it is reasonably easy to interpolate, as it grows monotonically and has a contiguous support (in contrast to the density function). Note that for multisets of values the CDF describes a step function. While the algorithms presented in this paper could directly interpolate this, we simplify the representation by disregarding the lower point of each step (i.e. connecting the upper points linearly). Disregarding the lower points of each step reduces the number of data points relevant for the interpolation, which speeds up the histogram construction, and already constructs a linear spline. Thus the goal of spline construction in this paper is to reduce a linear spline with n data points (n being the number of values in the original data) to a linear spline with k data points (k being the allowed histogram size) such that approximation is as good as possible.

3.2 Approximation Criterion

When approximating one function with another, it is necessary to define an approximation criterion to decide which approximation to choose. A commonly used measure is the mean squared error, like, for instance, used in [8] during the histogram construction. The mean squared error is very popular, but primarily because analytic solutions for minimizing it are known, the value itself is hard to interpret. Further, as only the average is minimized, the quality of the approximation can vary arbitrarily over the domain. As we want a smooth approximation, mean squared error is not a good choice.

A more robust error is the maximum error [13], which has a well defined meaning even for continuous domains. If we construct an approximation function with a maximum error of ϵ, we know that this maximum deviation of ϵ is guaranteed for all queries, regardless of the concrete value and without outliers. Of course this error measure is only helpful if we can achieve a reasonably low ϵ, we will see detailed algorithms for this in Section 4. For the spline interpolation, a maximum error ϵ defines an error corridor around the original function such that the resulting spline is guaranteed to stay within the error corridor. As a consequence, the maximum error is known over the whole domain. A slight variation of this is the maximum of the relative error, where the corridor size is defined relative to the real value. As we will later see, our algorithms can handle both relative and absolute errors. In our experiments we used the maximum relative error, as it is more critical in practice since.

3.3 Formalization

Using these observations, we can formalize the problem as follows: Given a linear spline S and a line segment $\overline{S_1 S_2}$, we define the maximum error induced by this line segment as the maximum deviation of knot points from the line segment. This is

$$\delta(S, \overline{S_1 S_2}) = \max_{p \in S \wedge p.x \in [S_1.x, S_2.x]} |p.y - \overline{S_1 S_2}[p.x]|$$

when using the absolute error, and

$$\delta(S, \overline{S_1 S_2}) = \max_{p \in S \wedge p.x \in [S_1.x, S_2.x]} \frac{|p.y - \overline{S_1 S_2}[p.x]|}{p.y}$$

when using the relative error. Note that it is sufficient to calculate the difference at the knot points, as the spline knots are connected linearly.

Using the maximum error induced by a single line segment, we can define the maximum error for interpolating a linear spline S by a linear spline S' as maximum of all errors induced by the line segments of S'.

$$\Delta(S, S') = \max_{1 \leq i < |S'|} \delta(S, \overline{S'[i] S'[i+1]})$$

Now the interpolation problem can be written as: Given a linear spline S and a natural number k, $k \geq 2$. Construct a linear spline S' such that $|S'| \leq k$, $S'[1] = S[1]$, $S'[|S'|] = S[|S|]$ and $\Delta(S, S')$ is minimal. Note that the requirement that the first and the last points of the splines must be identical prevents approximating splines that only cover part of the original spline.

4 Spline Construction

The goal of the spline construction algorithms is to construct a spline with a given size that approximates the cumulative frequency distribution with the minimal (or at least a low) maximum error. By using the spline interpretation of the original data as discussed in Section 3, the algorithms reduce a spline with n points (the original data) to a spline with k points. The key difficulty here is that n can be very large, potentially even larger than main memory, whereas k is relatively small (e.g. for $k = 100$ the histograms consume 1.6KB). Note that the spline representing the original data can be constructed trivially when the values are sorted and is therefore not an issue. We now present two algorithms for spline construction, first a greedy heuristic that can handle arbitrarily large inputs and then a DP based algorithm that constructs the optimal spline. For large inputs the two can be combined to produce good results, as we will see below.

4.1 Greedy Construction

Minimizing the maximum error with a greedy algorithm is difficult, as the effect of knot placement on later knots is hard to predict. Instead, we solve the dual problem (constructing the smallest spline with a given maximum error), and use its solution to solve the original approximation problem. The greedy spline construction is therefore split into two parts. The first part (*GreedySplineCorridor*) constructs a linear spline approximation for a given error corridor, and the second part (*GreedySpline*) minimizes the size of the error corridor using the first part.

```
GreedySplineCorridor
**Input:** a spline $S$, $|S| = n$ and an error corridor size $\epsilon$
**Output:** a spline connecting $S[1], S[n]$ through the corridor
$B = S[1], R =< B >$ // $S[1]$ is the first base point
$U = S[2] + \epsilon, L = S[2] - \epsilon$ // error corridor bounds
**for** $i = 3$ **to** $n$
  $C = S[i]$
  **if** $\overline{BC}$ is left of $\overline{BU}$ or right of $\overline{BL}$
    $B = S[i-1], R = R \circ < B >$
    $U = C + \epsilon, L = C - \epsilon$
  **else**
    $U' = C + \epsilon, L' = C - \epsilon$
    **if** $\overline{BU}$ is left of $\overline{BU'}$
      $U = U'$
    **if** $\overline{BL}$ is right of $\overline{BL'}$
      $L = L'$
$R = R \circ < S[n] >$
**return** $R$
```

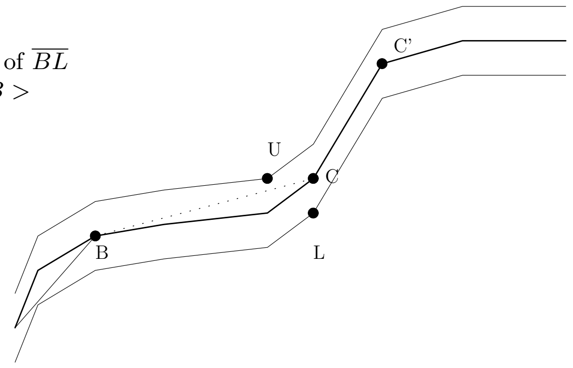


Fig. 1. Greedy Spline Approximation with a Given Error Corridor

The pseudo code of the spline construction with a given error corridor is shown in Figure 1. It starts constructing the result spline R by choosing the first point in S as first point of R. During the algorithm, the last point in R is called B (base point). It then performs a linear scan over the spline and checks if the line segment between B and the current point C is still within the error corridor. If not, the maximum error constraint would be violated if connecting B and C, i.e., $\delta(S, \overline{BC}) > \epsilon$. The last point before C allows for interpolation within the error boundaries, therefore it is chosen as new base point and added to R.

If a point is not chosen as a new base point, it still affects the error corridor. An important observation is that it is not required to keep track of the real error corridor, it is sufficient to remember the most limiting points. In the algorithm these are U for the the upper bound and L for the lower bound, that denote the most narrowing points of the error corridor relative to B. For each point C, the individual upper and lower bounds U' and L' are calculated by adding ϵ to the y value (or for relative errors by multiplying y with $1 + \epsilon$ and $1 - \epsilon$). If a new bound is more restrictive, it replaces the old bound. An illustration of this approach is included in Figure 1. The point B has been chosen as a base point (i.e. the second point in the output spline), and now the scan continues to point C. C is still reachable from B, thus it is skipped for now. The lower bound L is updated to the new lower bound derived from C, whereas the previous upper bound U is tighter than the bound derived from C and is thus unchanged. The next scan point C' is no longer reachable from B, therefore C will be added as new base point when scanning to C'.

The scan continues until the last point of S is reachable within the error corridor, after which it is added to R as the final point. The spline R now consists of the smallest spline (regarding the greedy selection method) that satisfies the maximum error ϵ across the domain. The whole algorithm performs a single scan over the original data and requires only constant memory in addition to the result itself. Thus, it has a runtime complexity of $O(n)$ and a space complexity of $O(k)$.

```
GreedySpline
**Input:** a spline $S$, $|S| = n$, and a desired spline size $k$
**Output:** a spline connecting $S[1], S[n]$ with $\leq k$ knots
$R = \emptyset, \epsilon_R = \infty$
$\epsilon_L = 0, \epsilon_U =$error when approximating $S$ with 0
while $\epsilon_U - \epsilon_L > \Delta$
  $\epsilon = \frac{\epsilon_L + \epsilon_U}{2}$
  $S_\epsilon =$GreedySplineCorridor($S$,$\epsilon$)
  if $|S_\epsilon| > k$
    $\epsilon_L = \epsilon$
  else
    if $\epsilon < \epsilon_R$
      $R = S_\epsilon$, $\epsilon_R = \epsilon$
    $\epsilon_U = \epsilon$
return $R$
```

Fig. 2. Greedy Spline Approximation Algorithm

This spline construction logic can now be used to find splines with a low maximum error. The pseudo code is shown in Figure 2: It performs a binary search over the maximum error until the desired accuracy is reached (which can be derived from the data). For each error bound ϵ, *GreedySplineCorridor* is used to construct a spline. If the spline is too large, the error bound has to be increased, otherwise it can be decreased. After the binary search is done, the algorithm returns the best spline found that consists of at most k points. In an implementation, the spline construction can stop if the spline candidate gets larger than k points, therefore the overall algorithm runs in $O(n \log n)$ time and $O(k)$ space. The algorithm is very fast, which makes it attractive as a pre-filter for slower algorithms (reducing a huge n to a more manageable n'). Due to its sequential access pattern it could even be used if the original spline does not fit into main memory.

4.2 Optimal Construction

Disregarding memory restrictions, there is no need to use an approximate solution, at least from a theoretical point of view. In his seminal work, Goodrich [15] showed that the optimal linear spline can be constructed in $O(n \log n)$, i.e., with the same asymptotic complexity as our greedy heuristic. Unfortunately these are primarily theoretical results. Several steps of the algorithm are very involved, and [15] already mentions that a potential implementation would have very large constants. To our knowledge, no implementation of the algorithm exists.

Instead, we use a dynamic programming algorithm that constructs the spline in $O(n^2k)$ time and $O(nk)$ space. The resulting spline is optimal when solely considering the original data points as potential spline knots, which is reasonable here (as $k \ll n$), but not necessarily optimal in some cases [15]. We ignore this possibility, as n is very large and offers enough potential knot points.

```
DPSpline
**Input:** a spline $S$, $|S| = n$ and a desired spline size $k$
**Output:** a spline connecting $S[1], S[n]$ with $\leq k$ knots
$dpPrevious$ = map $([1,n],[1,k]) \rightarrow [1,n]$
$dpError$ = map $([1,n],[1,k]) \rightarrow \mathbb{R}^+$
for $l = 1$ to $k$
  $dpError[1,l] = 0$, $dpPrevious[1,l] = 1$
for $i = 1$ to $n-1$
  for $j = i+1$ to $n$
    $\epsilon_{ij} = \delta(S, \overline{S[i]S[j]})$
    if $i = 1$
      $dpError[j,1] = \epsilon_{ij}$, $dpPrevious[j,1] = 1$
    for $l = 2$ to $k$
      $\epsilon = \max(\epsilon_{ij}, dpError[i,l-1])$
      if $i = 1$ or $\epsilon < dpError[j,l]$
        $dpError[j,l] = \epsilon$, $dpPrevious[j,l] = i$
$R = <>$, $r = n$
for $l = k$ to $1$
  $R =< S[r] > \circ R$
  $r = dpPrevious[r,l]$
return $R$
```

Fig. 3. Dynamic Programming Spline Construction Algorithm

The pseudo code for the DP algorithm is shown in Figure 3. It constructs a DP table that for each data point j $(1 \leq j \leq n)$ and each value l $(1 \leq l \leq k)$ gives the minimum error (*dpError*) of reaching this point with a spline of size l, and the previous point (*dpPrevious*) in the spline. The table is filled as follows: For the first point $S[1]$, the interpolation is perfect as we start with it, resulting in an error of zero for all spline sizes. Then, the algorithm considers all potential starting points of spline segments i and all potential ending points for each segment j. For each of these segment candidates it computes the maximum error ϵ_{ij} caused by choosing a spline segment $\overline{S[i]S[j]}$. Note that as the error corridor for the greedy algorithm, ϵ_{ij} can be maintained incrementally. If $i = 1$, i.e., the spline segment starts with the first point, we can construct a spline of size 1, setting $dpError[j,1] = \epsilon_{ij}$. Otherwise, the spline must consist of at least one other spline segment that reached i. The algorithm tries out all possible spline sizes l, using the maximum of ϵ_{ij} and the error of the preceding, smaller spline ($dpError[i,l-1]$) as total error ϵ. If ϵ is smaller than the best known error for this end point and this spline size, the error bounds are updated and the starting point stored in *dpPrevious*. After all segments were considered, $dpError[n,k]$ contains the maximum error of the best spline, and $dpPrevious[n,k]$ the previous to last point of this spline. Walking backwards over *dpPrevious*, the algorithm reconstructs the best spline R from the DP table.

4.3 Large Inputs

As the runtime complexity of the DP algorithm is $O(n^2k)$, it cannot be used for arbitrarily large problems. Our implementation could solve $n = 5,000$ and $k = 100$ in less than two seconds on a standard PC, but the runtime increases quadratically with n. And n can be very large in practice. We therefore use the greedy algorithm as a reduction step. The greedy algorithm itself is very fast even for large inputs ($n = 1,000,000$ in less than a second), and constructs approximating splines with known maximum errors. Thus, when solving large problems with the DP algorithm, we use the greedy algorithm to reduce the input to a spline with $5,000$ knots and then use the DP algorithm on the reduced spline. Due to the geometric interpretation of the error corridors, the errors of the two algorithms at most add up, and the error for $k = 5,000$ is very low. As we will see in Section 5, this allows us to efficiently handle very large inputs, while still maintaining hard error boundaries across the whole domain.

5 Experimental Results

In this section we study the prediction quality of our spline histograms in comparison with existing approaches. We first discuss the general setup, then we present the particular approaches that we consider in our evaluation, and finally we present the experimental results for real-world IR data.

5.1 General Setup

To evaluate the accuracy of the different approaches, we construct histograms of varying size for each approach. As the approaches differ in the information stored per bucket, we measure the size in bytes instead of buckets. For each histogram size, we construct 1 million range queries of the form $attr \geq x$, where x is varied uniformly between the minimum and the maximum attribute value. For each of these range queries, we compute the actual number of tuples in this range and the prediction made by each histogram. The difference between the two, the absolute error, is reduced by one to eliminate rounding issues (the predictions are reals while the actual number of tuples is an integer) and then divided by the actual number of tuples to get the relative error. We show three measures: the maximum relative error over all queries, as this was our optimization criterion and shows the worst case for each approach. The average relative error, as this show the “expected” error for an arbitrary query. And the mean squared error, as this is a standard measure in statistics.

Note that we (intentionally) use queries whose boundaries do not necessarily correspond to values occurring in the data. This mimics our original motivation, where the query compiler tries out potential constants, and in fact shows a major problem: Most approaches perform significantly better when only using range queries whose parameters occur in the data. Or in other words, most approaches fail when using arbitrary constants as range boundaries. This is often caused by a kind of overfitting, where the algorithms model the existing values very well at the expense of the general approximation. We include results when only considering existing values as comparison.

5.2 Considered Approaches

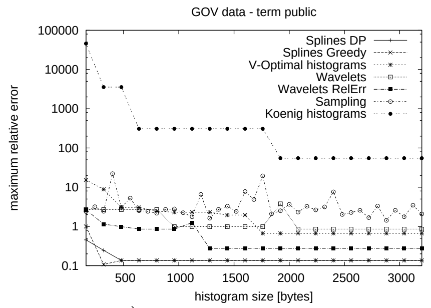
We compare our spline histograms with a number other approaches in the experiments. Our spline histograms, as described in Section 4, and are called *Greedy Spline* and *DP Spline* in the experiments. For the DP base spline construction, we first reduce the input to a spline with 5000 knots using the greedy construction and then run the DP algorithm to get the final spline. The *Sampling* strategy derives its estimation from a sample of the original data [16]. The *Koenig* histograms are described in [8], and construct a linear spline approximation within each histogram bucket. Unfortunately, they are only defined for integer domains in [8], and a generalization to real numbers is not straightforward (e.g., Formula 10 relies on the integer domain). As point queries are well defined even for real numbers, we used a CFG representation for range queries. The *V-Optimal* histograms described in [12] minimize the frequency variance in one bucket. This causes problems with continuous data, as most frequencies are 1. We therefore discretize the input into 2^{12} buckets. This number was derived empirically, changing the exponent reduced the accuracy. We also considered *Equi-Width* and *Equi-Depth* histograms, but omit the result here, as they are clearly dominated by *V-Optimal* histograms. The *Wavelet* histograms are described in [14]. They require a discrete domain, therefore we discretize the values into 2^{10} buckets (again, empirically determined). The *Wavelet RelErr* histograms described in [13] use a different coefficient selection algorithm to minimize the relative error, we therefore include them in our experiments.

5.3 Real Data

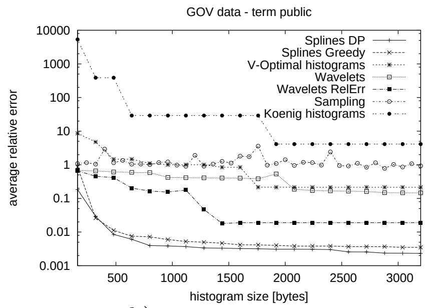
As a data set we use the GOV collection from the TREC-12 Web Track benchmark (http://trec.nist.gov/), which consists of roughly 1.25 million documents crawled from the .gov domain. We compute the inverted lists for the terms occurring in the Web Track's topic distillation task, scoring each document with normalized TF*IDF*PageRank scores (real values in $[0, 1]$). We show here the results of the inverted list for term *public*, which is the largest inverted list (the other terms show similar results). The inverted lists contains of $427,940$ entries with $359,505$ distinct score values. The scores themselves are heavily skewed towards low values, even though there are relatively few duplicates.

Figure 4 a shows the maximum relative error of the different approaches with varying histogram size. As the maximum is sensitive to outliers, the maximum error is quite high for most approaches. The spline histograms guarantee a maximum error of $\approx 13\%$ for all queries. Interestingly even this number is a kind of outlier, as it is caused by ignoring the steps in the distribution function in our data model. A more forgiving error measure is the average relative error, as this shows the expected error for an arbitrary query. The results are shown in Figure 4 b. Again, the spline histograms perform very well, with an average error of $\approx 0.3\%$ for *Greedy Splines*. The *DP Splines* perform a bit better ($\approx 0.2\%$), but the overall performance is similar.

A very popular error measure in statistics is the mean squared error, which we show in Figure 5 a. The interpretation of the absolute values is somewhat difficult, as the impact on queries is unclear, but low values are most likely

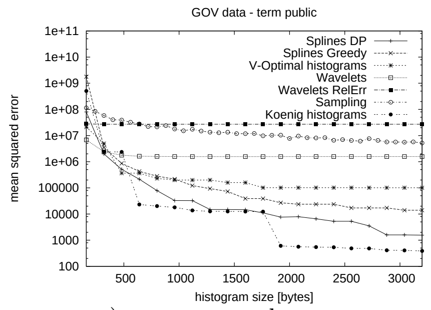


a) maximum error



b) average error

Fig. 4. Maximum and average relative error for the GOV term list *public*



a) mean squared error

	max rel.	avg rel.	mean sq.
DP Spline	0.0096	0.0008	50,029
Greedy Spline	0.0040	0.0024	497,503
V-Optimal	0.4880	0.0034	162,246
Wavelet	0.8571	0.0244	$5.4 * 10^7$
Wavelet RelErr	0.2762	0.1001	$8.7 * 10^8$
Koenig	55.068	0.0005	435
Sampling	215,012	4.49839	$1.1 * 10^8$

b) querying for existing data points (3200 bytes)

Fig. 5. Mean squared error and querying for existing data for the GOV term list *public*

good. Again, the spline histograms perform very well, only Koenig histograms which explicitly minimize the mean squared error perform better. Note though that Koenig histograms performed quite poorly for the more insightful relative error. Some of approaches, in particular Koenig histograms, perform surprisingly badly in the experiments seen so far. This is due to the fact that the query boundaries are chosen from all over the domain, regardless of whether a data item with this exact value exists or not. If we instead chose all existing values as query boundaries once, the results are much better (Figure 5 b), which suggests overfitting to the data. Koenig histograms for example perform very well when only queried for existing values, but perform quite poor overall. The DP strategy has a slightly higher maximum error than the Greedy strategy. This is caused by the pre-filtering steps, which removes data points from its input, causing an underestimation of the maximum error.

5.4 Effect on Queries

While the accuracy results above compare histograms using different error metrics, we measure the effect on real queries in this section. We use the optimization framework described in [6], as the experiments in the paper operate on the same data set, using the official GOV benchmark queries. For the comparison, we optimize the queries using different cardinality estimators, execute them, and then compare the resulting runtime and network traffic. Due to space constraints, we only show the average results for the *Spline* histograms (Greedy and DP produced the same plans), *V-Optimal* histograms, and *Wavelet RelErr* histograms, as these were by far the most accurate:

	Splines	V-Optimal	Wavelet
runtime [ms]	585	649	618
net [bytes]	23,918	39,660	26,715

Here, even the relatively simple GOV queries can be improved just by using splines instead of the other histograms. Note that using the *WaveLet RelErr* histograms increased the optimization time to > 10 min and the memory consumption to $> 2.5GB$, as the optimization framework constructs histograms over intermediate result approximations and the wavelet construction is very expensive. Optimization with the other histograms took < 20 ms.

In conclusion, our spline histograms offer smooth predictions for arbitrary values, and guarantee a known maximum error over the whole domain.

References

1. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. **66**(4) (2003) 614–656
2. Güntzer, U., Balke, W.T., Kießling, W.: Optimizing multi-feature queries for image databases. In: VLDB. (2000) 419–428
3. Nepal, S., Ramakrishna, M.V.: Query processing issues in image (multimedia) databases. In: ICDE. (1999) 22–29
4. Cao, P., Wang, Z.: Efficient top-k query calculation in distributed networks. In: PODC. (2004) 206–215
5. Yu, H., Li, H.G., Wu, P., Agrawal, D., Abbadi, A.E.: Efficient processing of distributed top-k queries. In: DEXA. (2005) 65–74
6. Neumann, T., Michel, S.: Algebraic query optimization for distributed top-k queries. In: BTW. (2007) 324–343
7. Michel, S., Triantafillou, P., Weikum, G.: Klee: A framework for distributed top-k query algorithms. In: VLDB. (2005) 637–648
8. König, A.C., Weikum, G.: Combining histograms and parametric curve fitting for feedback-driven query result-size estimation. In: VLDB. (1999) 423–434
9. Ioannidis, Y.E.: The history of histograms (abridged). In: VLDB. (2003) 19–30
10. Deshpande, A., Garofalakis, M.N., Rastogi, R.: Independence is good: Dependency-based histogram synopses for high-dimensional data. In: SIGMOD. (2001) 199–210
11. Poosala, V., Ioannidis, Y.E., Haas, P.J., Shekita, E.J.: Improved histograms for selectivity estimation of range predicates. In: SIGMOD. (1996) 294–305
12. Jagadish, H.V., Koudas, N., Muthukrishnan, S., Poosala, V., Sevcik, K.C., Suel, T.: Optimal histograms with quality guarantees. In: VLDB. (1998) 275–286
13. Garofalakis, M.N., Kumar, A.: Wavelet synopses for general error metrics. ACM Trans. Database Syst. **30**(4) (2005) 888–928
14. Matias, Y., Vitter, J.S., Wang, M.: Wavelet-based histograms for selectivity estimation. In: SIGMOD. (1998) 448–459
15. Goodrich, M.T.: Efficient piecewise-linear function approximation using the uniform metric. Discrete & Computational Geometry **14**(4) (1995) 445–462
16. Scott, D.W.: Multivariate Density Estimation: Theory, practice, and visualization. Wiley (1992)