Optimal Histograms with Outliers

Rachel Behar
The School of Computer Science & Engineering
Jerusalem, Israel
rachel.beharharr@mail.huji.ac.il

Sara Cohen
The School of Computer Science & Engineering
Jerusalem, Israel
sara@cs.huji.ac.il

ABSTRACT

Histograms are a well studied and simple way to summarize data. As such, they are used extensively in a variety of applications that require estimates of data frequency values. Significant previous work has studied the problem of finding optimal histograms with respect to an error measure. In this paper we study the classic problem of finding an optimal histogram for a dataset, with a new twist: The histogram must contain at least $n - k$ of the n data points. The k excluded data points are considered outliers.

We consider two notions of excluding data items, by allowing arbitrary items to be excluded, or only removing items while retaining a *consistent* histogram. Polynomial algorithms are presented for these problems. Significant experimentation demonstrates that our algorithms work well in practice to reduce the histogram error.

1 INTRODUCTION

Covering a set of data points with a histogram is a fundamental problem that lays at the heart of many database applications. Problems of this sort appear as optimization problems with various constraints on the type and size of the histogram, as well as an error function that must be minimized. Among these problems, finding an optimal histogram covering all but a few input points is of interest in view of outlier removal. In such a problem, given n data points, we are asked to find the optimal histogram covering at least $n - k$ of the n input points. From the viewpoint of optimization, excluding k points reduces the error and in this sense, the excluded data points can be considered outliers. In computational geometry, variations of this problem have been studied in many different settings.

A histogram provides the user with an estimate of frequencies of data elements within each bucket. For example, consider a histogram over integer exam grades. The bucket $([51, 60]; 40)$ indicates that there are 40 exam grades between 51 and 60, inclusive, in the dataset. Without additional information, a uniform distribution of elements within the bucket will be assumed, i.e., 4 grades each for every number between 51 and 60. In practice, this may be far from the correct frequencies, e.g., it is possible that there are 19 occurrences each of grade 59 and 60 and two occurrences of grade 51.

Using an error metric, such as sum-squared error, it is possible to measure the distance between the histogram's estimation and the actual frequencies. When the error is large, the histogram poorly captures the data. The simplest way to decrease the error is to increase the number of buckets. By using more buckets, the user has a more fine-grained view of the data. This, however, is not always an appropriate solution, and in particular, is inappropriate if the histogram is given to a human to view (and not simply to a computer program to continue manipulating for some purpose). If the user is given a very large number of buckets she loses the ability, to some degree, of seeing "the big picture" of the data. It is more difficult to grasp the meaning of the data when many buckets are displayed. A large number of buckets also degrades the quality of the user interface, particularly on mobile devices.

Histograms with a limited number of buckets can have high error as they must cover all elements within the dataset. However, there can be elements that are outliers, in the sense that it is their inclusion that causes the error to increase. In the above example of a dataset of grades, the occurrences of grade 51 can be seen as outliers. Indeed, if these elements did not have to be covered by a bucket, one could use $([59, 60]; 38)$, for which the uniform distribution perfectly estimates the actual grades in the bucket range.

In this paper our goal is to find optimal histograms, given a bound β on the number of buckets, as well as a bound k on the number of outliers that need not be covered. In other words, the goal is to optimally choose up to k elements and a histogram with up to β buckets, so as to minimize the total error.

The main contributions of the paper are:

- We introduce two intuitive notions for an optimal histogram (called a *summary*) with deletions.
- We present several algorithms for solving this problem, including algorithms that are provably optimal.
- Extensive experimentation demonstrates the quality of our solutions.

This paper is organized as follows. We start by discussing related work in Section 2. In Section 3 we set up the framework by providing the necessary definitions. In Sections 4 and 5 we study the optimal summary with deletion problem when there is only a single bucket, or when multiple buckets are allowed. In Section 6 we consider datasets with multiple columns. Extensive experimentation appears in Section 7, and Section 8 concludes.

2 RELATED WORK

Related work falls into two sub-areas: finding optimal histograms and identifying outliers in data. We discuss each of these aspects below.

Histograms. Histograms summarize data by storing the number of values within a given data range. They are traditionally used for estimating costs of query plans, as well as for approximate query answering. Various types of histograms have been considered in the past. These differ both on their maintenance efficiency over dynamic data, and their effectiveness for approximating query answers. Some of the classical types of histograms considered include equi-width (in which all buckets have the same range size), equi-depth (in which all buckets have the same number of data elements), serial (which group elements according to frequency) and end-biased (in which all buckets except one have a single element range). This paper considers the popular v-optimal histograms [9] which has been proven to be optimal

in minimizing the variance of element frequencies within the buckets. A taxonomy of histogram types was presented in [12]

Due to their popularity, there has been much effort to efficiently compute histograms from data. In [11], a polynomial time algorithm was presented to compute v-optimal histograms. This work was later expanded in [10] for the case in which there is a bound on the total number of buckets that can be used to summarize several different columns of data. There has also been work on approximating histograms, e.g., [7, 8], and using sampling for constructing histograms [5].

No previous work has considered the setting of computing histograms when not all of the elements must be covered. Among previous work, our paper is most related to [11] as we also present an optimal algorithm for histogram creation, albeit in a new setting.

Outliers. Intuitively, an outlier is a data element that is distant from other elements. Many different criteria have been considered in the past to determine when a data point is an outlier, and outlier detection algorithms have been presented. In particular, algorithms differ as to whether they compare data elements with the entire set, or only with a small local subset. In our work, we assume that the user provides us with a number k of outliers that can be removed. This differs from most previous work, in which no such number is provided. In [4, 14] algorithms for top-k outliers are presented. However, they may return less than k elements if, fewer are deemed to be outliers. (In addition, they differ in their notion of an outlier, as they do not attempt to minimize histogram error.)

Histograms have been used to efficiently find outliers [6, 13]. Our algorithms can also be viewed as finding outliers using histograms. However, our approach is significantly different from previous work. In [6, 13], histograms are built over the entire data set, and then data elements that significantly differ from their fellow bucket elements are considered outliers. In our work, we find the optimal set of data elements that should be removed in order to generate a histogram with low error.

Several works have considered similar types of problems, i.e., covering all but k data elements in an optimal manner (in some sense). In particular, [2] considers optimality in terms of minimum diameter, minimum area bounding box or minimal area convex hull. In [3], the goal is to optimally remove k data elements so that the remaining have a minimum-width square or rectangular annulus. Covering all but k data elements with disjoint boxes is considered in [1]. While these works have a similar goal, they differ in the type of cover that must be created and in error function that should be minimized. Hence, previous techniques cannot carry over to our framework.

3 DEFINITIONS

Datasets and Frequencies. The goal of this paper is to find a method to effectively summarize a large dataset, such as the contents of a relation in a database, or the result of a query. In the following, a *dataset* P is a multiset of elements. To simplify the presentation, we assume that the domain of P is $\mathbb{Z}$. However, all our results immediately generalize to arbitrary completely ordered discrete domains.[1]

Given a multiset of elements P, for all $q \in \mathbb{Z}$, we use $card(q, P)$ to denote the number of copies of q that appear in P. We use $\vec{f}_P$ to denote the *frequency vector* of P, i.e., $\vec{f}_P[q] = card(q, P)$. Note that $\vec{f}_P$ has finitely many non-zero elements. We differentiate between the *elements* of $\vec{f}_P$ denoted p, q, r, and the *frequency values* (or simply *values* for short) of $\vec{f}_P$, denoted v, w.

Example 3.1. Consider the multiset, used as running example,

$$P = \{\!\{10, 20, 30, 20, 30, 40, 10, 40, 50, 0, 0, 0, 0\}\!\}.$$

The frequency vector of P, when considering only non-zero elements, is

$$\vec{f}_P[q] = \{0{:}4, 10{:}2, 20{:}2, 30{:}2, 40{:}2, 50{:}1\} \quad \square$$

Summaries. A *bucket* is written $b = ([p, q]; n)$, where

- $p \le q$ are integers defining the bucket *endpoints* and
- n is a *positive* natural number indicating the number of elements in the bucket.

We say that $\{p, \ldots, q\}$ is the range of b.

Now, consider a multiset P and a bucket $b = ([p, q]; n)$. We say that b is *consistent* with P if $\sum_{r=p}^{q} \vec{f}_P[r] = n$, i.e., n is precisely the sum of frequencies of all elements within the range of b in P.

We say that a set of buckets B is a *summary* of P if

- every two buckets in B have disjoint ranges,
- all buckets in B are consistent with P,
- for every $r \in P$, there is some $b \in B$ such that r is in the range of b.

Thus, intuitively, a summary of P is a set of disjoint consistent buckets that cover all elements appearing in P.

Example 3.2. The following are different summaries of dataset P from Example 3.1, with one and two buckets, respectively:

$$B_1 = \{([0, 50]; 13)\} \qquad B_2 = \{([0, 0]; 4), ([1, 50]; 9)\} \quad \square$$

Remark 1. *A summary of a multiset of elements is very similar to the well-studied notion of a histogram of a multiset of elements. Much of the previous work on histograms assumed that the buckets fully covered the domain. This was important, as data was dynamically added to the dataset and had to be incorporated into the histogram (and therefore, had to have a bucket to which it could be added). In this work, since our goal is to summarize the current contents of a dataset, we only create non-empty buckets, with endpoints corresponding to actual elements in the dataset. Empty buckets are not of interest.*

We associate a summary B with a vector $\vec{e}_B$ used to *estimate* frequencies of elements, as follows. Let r be an integer. If r is within the range of a bucket $([p, q]; n) \in B$, then $\vec{e}_B[r] = \frac{n}{q-p+1}$. Otherwise, $\vec{e}_B[r] = 0$.

Now, the *error* of B w.r.t. P, denoted $err(B, P)$ is the distance of $\vec{e}_B$ from $\vec{f}_P$. In general, different distance metrics can be employed. In this paper, we use the sum-squared-error (SSE) metric as, used by [11]:

$$err(B, P) = \sum_{q \in \mathbb{Z}} (\vec{f}_P[q] - \vec{e}_B[q])^2 .$$

However, the work generalizes to additional metrics. It has been shown [11] that this error metric can be equivalently computed as:

$$err(B, P) = \sum_{([p,q];n) \in B} \left(\sum_{p \le r \le q} (\vec{f}_P[r])^2 - \frac{n^2}{q - p + 1} \right). \quad (1)$$

Example 3.3. Recall B_1 and B_2 from Example 3.2. Then, $err(B_1, P) = 29.7$ and $err(B_2, P) = 15$. $\square$

[1] This work considers only discrete domains. For attributes over continuous domains, our work can be applied if a discretization function is used, e.g., by rounding to a given decimal place.

Optimal Summaries. For any dataset P, it is possible to find a summary B with $err(B, P) = 0$ by simply creating a singleton bucket for each element appearing in P. However, such a summary will be of size similar to that of P itself, and will be unwieldy for large datasets. Hence, we will be interested in creating summaries of bounded size.

Let β be a positive integer, called the *bucket size bound*. A summary B is size-bounded by β if the number of buckets in B is at most β. Given a dataset P and a bucket size bound β, the *optimal summary problem* is to find a summary B of P that is size-bounded by β, and has minimal error. This problem has been considered in the past, and has been shown to be solvable in polynomial time using dynamic programming [11].

Optimal Summaries with Deletion. In practice, some of the elements in the dataset may be outliers that greatly contribute to the error of the summary, while adding little information of interest. When we are interested in summarizing the data to understand the general trends, such outliers may not be of interest. Indeed, a better understanding of the data may be derived when some elements are removed, to derive a smoother dataset, before creating a summary.

We say that P' is a *k-deletion* of P if P' is derived by removing at most k elements from P. We consider two different types of summaries after deletion. We say that B is a *k-deletion summary* of P if there is a k-deletion P' of P such that B is a summary of P'. A stronger requirement on the summary can be formulated as follows: We say that B is a *consistent k-deletion summary* of P if there is a k-deletion P' of P such that B is a summary of P' *and* every bucket in B is consistent with P.

Example 3.4. Consider the dataset

$$P = \{\!\{1, 1, 2, 2, 2, 3, 3\}\!\}.$$

Suppose that we can only use a single bucket, but we may delete up to $k = 2$ elements.

The optimal choice is to delete a single occurrence of element 2, yielding the dataset $P' = \{\!\{1, 1, 2, 2, 3, 3\}\!\}$, which can be summarized using one bucket with an error of 0. This summary $B' = ([1, 3]; 6)$, however, is not consistent with P.

An optimal consistent 2-deletion summary can be derived by removing all occurrences of the element 1 (or all occurrences of the element 3) yielding the consistent summary $B'' = ([2, 3]; 5)$ with error 0.5.

This example also serves to demonstrate why consistent summaries have an added benefit. The summary B'' accurately reflects the dataset, namely that it contains five elements in the range $[2, 3]$. The summary B' can be seen as being somewhat confusing, as it seems to imply that the dataset has six elements in the range $[1, 3]$, when in fact there are seven such elements (and one occurrence of element 2 has been removed to reduce the error). □

Problems of Interest. In this paper we study the following problem, called the (consistent) optimization problem: *Let P be a dataset, β be a bucket size bound and k be a natural number. Find P', B such that P' is a k-deletion of P and B is a summary of P' (and B is consistent with P) such that $err(B, P')$ is minimized.*

4 SINGLE BUCKET SUMMARIES

Before studying the general optimization problems, we start by assuming that only a single bucket may be used. We note that even in this case there can be exponentially many different choices of k elements to delete. Hence, finding an optimal k-deletion summary with one bucket is not a trivial problem. In addition, the ability to optimize for a single bucket is an important component in a solution for multiple buckets. Therefore, in this section we present algorithms that find an optimal k-deletion summary in polynomial time, when only a single bucket can be used. We start by considering arbitrary single bucket summaries with deletions, and then consider the special case of consistent single bucket summaries with deletions.

4.1 Arbitrary Single Bucket Summaries

We use $minArbErr(\vec{f}, \beta, k)$ to denote the minimal error that can be achieved for a multiset with frequency vector $\vec{f}$, when up to k elements can be deleted from the multiset, and up to β buckets can be used. The algorithms that we will present compute this error value directly. As is standard with dynamic programming, to find the k elements to be deleted from the multiset and the bucket boundaries that should be used, some additional book-keeping is necessary. This is quite straight-forward, but clutters the presentation. Hence, we focus on computing $minArbErr(\vec{f}, \beta, k)$.

In this section, we study the problem of computing the value $minArbErr(\vec{f}, 1, k)$ for a given $\vec{f}$ and k, i.e., our optimization problem for the special case where only one bucket can be used. Intuitively, there are two different ways to choose elements to remove, in order to reduce the error:

(1) we can remove the first or last elements with non-zero frequency in $\vec{f}$, thereby allowing a bucket with smaller range to cover $\vec{f}$;
(2) we can remove elements that have the greatest frequency, thereby reducing the variance of the frequencies within the bucket.

Before presenting our algorithm, we introduce some necessary notion used in the algorithms throughout this paper. Let $\vec{f}$ be a frequency vector. We use $\text{size}(\vec{f})$ to denote the number of different non-zero elements in $\vec{f}$, and $\text{first}(\vec{f})$ and $\text{last}(\vec{f})$ to denote the first and last elements with non-zero frequency in $\vec{f}$, respectively. Finally, $\text{next}(\vec{f}, p)$ and $\text{prev}(\vec{f}, p)$ denote the element immediately after and immediately preceding p in $\vec{f}$ with non-zero frequency, respectively. When $\vec{f}$ is clear from the context, it will be omitted, and we will simply write size, first, last, $\text{prev}(p)$, $\text{next}(p)$.

For elements p, q, let $\vec{f}_{[p,q]}$ be the frequency vector

$$\vec{f}_{[p,q]}[r] = \begin{cases} \vec{f}[r] & p \le r \le q \\ 0 & \text{otherwise} \end{cases}$$

The algorithm ArbSingleBErr in Figure 1 computes the value $minArbErr(\vec{f}, 1, k)$. ArbSingleBErr starts by first checking if all elements in the vector can be removed (Lines 1–2), in which case an error of zero is returned. Next, the algorithm considers all options of removing first or last elements, as well as other elements with high frequency. Intuitively, the values k_{left} and k_{right} in the algorithm are used to specify how many elements can be removed from the beginning and end, respectively, of $\vec{f}$. This leaves $k_{mid} = k - k_{left} - k_{right}$ elements of high frequency to be removed from the remainder of $\vec{f}$.

The algorithm ArbSingleBErr therefore iterates over values of $k_{left} \le k$ (Line 6), and values of $k_{right} \le k - k_{left}$ (Line 9). We note that we always choose values for k_{left} and k_{right} that are sufficient to reduce the frequency of elements at the beginning

Algorithm ArbSingleBErr($\vec{f}, k$)

1. **if** $\sum_{p=\text{first}}^{\text{last}} \vec{f}[p] \leq k$
2. **then return** 0
3. $e_* \leftarrow \infty$
4. $p \leftarrow$ first
5. $k_{left} \leftarrow 0$
6. **while** $k_{left} \leq k$
7. **do** $q \leftarrow$ last
8. $k_{right} \leftarrow 0$
9. **while** $k_{right} \leq k - k_{left}$
10. **do** $k_{mid} \leftarrow k - k_{left} - k_{right}$
11. $e \leftarrow \text{LowerMaxError}(\vec{f}_{[p,q]}, k_{mid})$
12. $e_* \leftarrow \min\{e_*, e\}$
13. $k_{right} \leftarrow k_{right} + \vec{f}[q]$
14. $q \leftarrow \text{prev}(q)$
15. $k_{left} \leftarrow k_{left} + \vec{f}[p]$
16. $p \leftarrow \text{next}(p)$
17. **return** e_*

Algorithm LowerMaxError($\vec{f}, k$)

1. $p_{min}{:}v_{min} \leftarrow \text{minFreqVal}(\vec{f})$
2. $p_{max}{:}v_{max} \leftarrow \text{maxFreqVal}(\vec{f})$
3. **while** ($k > 0$ and $v_{max} > v_{min}$)
4. **do** $\vec{f}[p_{max}] \leftarrow \vec{f}[p_{max}] - 1$
5. $k \leftarrow k - 1$
6. $p_{max}{:}v_{max} \leftarrow \text{maxFreqVal}(\vec{f})$
7. $n_{\text{sum}} \leftarrow \sum_{p=\text{first}}^{\text{last}} \vec{f}[p]$
8. $n_{\text{sqsum}} \leftarrow \sum_{p=\text{first}}^{\text{last}} (\vec{f}[p])^2$
9. **return** $n_{\text{sqsum}} - {n_{\text{sum}}}^2/(\text{last} - \text{first} + 1)$

Figure 1: Returns $minArbErr(\vec{f}, 1, k)$**.**

and end of $\vec{f}$ to zero. This is achieved by increasing k_{left} and k_{right} by the actual frequency values appearing in $\vec{f}$ (Lines 15 and 13).

For given values of k_{left} and k_{right} we have p and q that correspond to the first and last elements of $\vec{f}$ whose frequency has not been reduced. They determine the range of values in $\vec{f}$ for which we will attempt to reduce the highest frequency values (to reduce variance) using the algorithm LowerMaxError.

LowerMaxError uses minFreqVal (resp. maxFreqVal) to return the element and corresponding frequency value which is minimal (resp. maximal) in $\vec{f}$. While more elements can be removed ($k > 0$) and there is still non-zero variance within the range ($v_{max} > v_{min}$) the frequency of the most frequent element is reduced by one (Line 4). Finally, Line 9 returns the error for the single bucket that would be created for the updated $\vec{f}$, using Equation 1. ArbSingleBErr keeps track of the lowest error option (Line 12), and returns this value (Line 17).

Correctness and runtime of ArbSingleBErr are stated in the following theorem.

Theorem 4.1. *For a given $\vec{f}$ and k,* ArbSingleBErr($\vec{f}, k$) *computes $minArbErr(\vec{f}, 1, k)$ in time $O(k^2(\text{size}\log\text{size} + k))$.*

Proof. ArbSingleBErr iterates over all options of removing all occurrences of elements (i.e., reducing to a zero frequency) from the beginning and end of $\vec{f}$. For each such option, LowerMaxError is used to reduce the frequencies of highest frequency elements. Hence, it is sufficient to show that LowerMaxError returns the minimal error that can be derived by k-deletions that do not remove all occurrences of the first and last elements of $\vec{f}$.

In the following, assume that p and q are the first and last elements, respectively, with non-zero frequency in $\vec{f}$. Thus, the single bucket over a k-deletion of $\vec{f}$ will have the form $([p, q]; n)$ where n is the sum of frequencies of the remaining elements of $\vec{f}$.

Recall that LowerMaxError does the following simple action: while there are still elements that can be deleted and not all frequencies in $\vec{f}$ (between p and q) are equal, find the most frequent element, and reduce its frequency by one. We will prove that this strategy achieves minimal error.

Assume, by way of contradiction, that there exist a different strategy to delete up to k elements from $\vec{f}$, that achieves the minimal error e_0. Let $\vec{f}'$ be this k-deletion. Let

$$v_{est} = (\sum_{p \leq r \leq q} \vec{f}'[r])/(q - p + 1).$$

Note that v_{est} is the estimated frequency value for each element in the bucket. The error of $\vec{f}'$ is

$$e_0 = \sum_{p \leq r \leq q} (\vec{f}'[r] - v_{est})^2 .$$

We will show that LowerMaxError also returns e_0.

Let $v_- = \min_{p \leq r \leq q}\{\vec{f}'[r] \mid \vec{f}'[r] \neq \vec{f}[r]\}$, i.e., v_- is the minimal frequency in $\vec{f}'$ changed by the deletion. Let $v_+ = \max_{p \leq r \leq q}\{\vec{f}'[r]\}$, i.e., v_+ is the maximal frequency value in $\vec{f}'$.

Now we consider the frequency vector derived by reducing the frequency v_+ by one and increasing the frequency v_- by one. The error of this new frequency vector e_0' differs from e_0 only in the summation for frequencies v_+ and v_-. Now we will show that $e_0' - e_0 \leq 0$ i.e, this change did not increase the error. Observe that

$$\begin{aligned} e_0' - e_0 &= (v_+ - 1 - v_{est})^2 + (v_- + 1 - v_{est})^2 - \\ &\quad \left((v_+ - v_{est})^2 + (v_- - v_{est})^2\right) && (2) \\ &= 2(v_- - v_{est}) - 2(v_+ - v_{est}) + 2 && (3) \\ &\leq 2(v_- - v_{est}) - 2(v_- + 1 - v_{est}) + 2 = 0 && (4) \end{aligned}$$

where Equation 4 holds since $v_- + 1 \leq v_+$, since otherwise we could have produced $\vec{f}'$ using LowerMaxError.

Now, by repeating this change up to k times, the result of LowerMaxError will be equal to the minimum error.

To prove the runtime, observe that ArbSingleBErr makes at most k^2 calls to LowerMaxError. Each call to LowerMaxError costs size log size + k. To achieve this time, we first sort the vector $\vec{f}_{[p,q]}$ sent to the algorithm. This allows us to efficiently evaluate all calls to minFreqVal and maxFreqVal. Then, we require another k operations to lower the frequencies, giving a total of $O(\text{size}\log\text{size} + k)$ for algorithm LowerMaxError, and $O(k^2(\text{size}\log\text{size} + k))$ for ArbSingleBErr, as required. □

4.2 Consistent Single Bucket Summaries

We use $minConErr(\vec{f}, \beta, k)$ to denote the minimal error that can be achieved for a multiset with frequency vector $\vec{f}$, when a k-deletion is taken and up to β buckets can be used, and the summary created *must be consistent* with $\vec{f}$. In this section we show how to compute this value for the special case where $\beta = 1$.

ConSingleBErr (Figure 2) uses a precomputed matrix SSE of dimensions size($\vec{f}$)×size($\vec{f}$), where $SSE[p, q]$ is the sum-squared error for a single bucket over the range $[p, q]$ of $\vec{f}$, with no

Algorithm ConSingleBErr($\vec{f}, k$)

1. **if** $\sum_{p=\text{first}}^{\text{last}} \vec{f}[p] \leq k$
2. **then return** 0
3. $q \leftarrow$ last
4. $k_{right} \leftarrow 0$
5. **while** $k_{right} + \vec{f}[q] \leq k$
6. **do** $k_{right} \leftarrow k_{right} + \vec{f}[q]$
7. $q \leftarrow \text{prev}(q)$
8. $e \leftarrow SSE[\text{first}, q]$
9. $p \leftarrow$ first
10. $k_{left} \leftarrow 0$
11. **while** $k_{left} + \vec{f}[p] \leq k$
12. **do** $k_{left} \leftarrow k_{left} + \vec{f}[p]$
13. $p \leftarrow \text{next}(p)$
14. **while** $k_{left} + k_{right} > k$
15. **do** $q \leftarrow \text{next}(q)$
16. $k_{right} \leftarrow k_{right} - \vec{f}[q]$
17. $e \leftarrow \min\{e, SSE[p, q]]\}$
18. **return** e

Figure 2: Returns $minConErr(\vec{f}, 1, k)$.

deletions,[2] i.e., $SSE[p, q] = \sum_{p \leq r \leq q} (\vec{f}[r] - v_{est})^2$ where $v_{est} = (\sum_{p \leq r \leq q} \vec{f}[r])/(q - p + 1)$.

The algorithm ConSingleBErr first checks if all elements in $\vec{f}$ can be deleted. In this case, the error is zero (Lines 1–2). Otherwise, it begins by considering the case in which elements on the right-hand side of $\vec{f}$ are maximally deleted and computes the error in this case (Lines 5–8). Then, in Lines 11–17 we slowly increase the number of elements removed on the left-hand side, while reducing the number of elements removed on the right-hand side, so as to remain within the limit of k. For each such option, we compute the error, and keep the minimal value.

We show the following result.

Theorem 4.2. *For a given $\vec{f}$ and k,* ConSingleBError($\vec{f}, k$) *computes $minConErr(\vec{f}, 1, k)$ in time $O(\min\{k, \text{size}\})$.*

Proof. To prove correctness, it is important to note that only maximal deletions from the two sides of $\vec{f}$ need to be considered. Thus, for example, in Lines 5–7 we maximally remove elements from the right-hand side of $\vec{f}$. This follows from the following claim: Let $\vec{f}$ be a frequency vector, and $p < q$ be elements with non-zero frequencies in $\vec{f}$. Let n be the total frequency of all elements in $\vec{f}$ and let n' be the total frequency of all elements between p and q (including) in $\vec{f}$. Then, $err(\{([\text{first}, \text{last}]; n)\}, \vec{f}) \geq err(\{([p, q]; n')\}, \vec{f}_{[p,q]})$. This follows from Lemma 2 in [11].

Finally, to show the runtime, observe that we iterate over $\vec{f}$ twice: once from right to left (Lines 5–7) and once from left to right (Lines 11-17). Thus, the time is at most $O(\text{size})$. Since in both our iterations over $\vec{f}$ we do at most k operations (we increase k_{right} or k_{left} until k), the time is at most $O(k)$. Together, this proves our claim of runtime. □

[2] In practice, it is not necessary to store a matrix of size size × size. Instead this value can be computed in time $O(1)$ given two arrays of size size each, as discussed in [11].

Algorithm NoDeletionSummary($\vec{f}, \beta$)

1. **for** $p \leftarrow$ first to last
2. **do** $M[p, 1] \leftarrow SSE[\text{first}, p]$
3. **for** $\beta' \leftarrow 2$ to β
4. **do for** $q \leftarrow$ next(first) to last
5. **do** $M[q, \beta'] \leftarrow \infty$
6. **for** $p \leftarrow$ prev(q) down-to first
7. **do** $e \leftarrow M[p, \beta' - 1] + SSE[\text{next}(p), q]$
8. $M[q, \beta'] \leftarrow \min\{M[q, \beta'], e\}$
9. **return** $M[\text{last}, \beta]$

Figure 3: Returns $minArbErr(\vec{f}, \beta, 0)$ (which is also equal to $minConErr(\vec{f}, \beta, 0)$).

5 MULTI-BUCKET SUMMARIES

We now consider the problem of finding summaries that can have multiple buckets. The problem of finding an optimal summary, when no deletions are permitted ($k = 0$) is well studied. Hence, as a baseline approach it is natural to try to directly leverage algorithms that find an optimal summary, in order to solve our problem.

5.1 Baseline Approaches

Before discussing how to find summaries with deletions, we review the dynamic algorithm proposed in [11] for building optimal sum-squared-error histograms without deletions.

Optimal Summary without Deletions. Figure 3 presents the algorithm NoDeletionSummary from [11] which computes the optimal error for a multiset with frequency vector $\vec{f}$ when using β buckets but no deletions. Note that in this special case, clearly $minArbErr(\vec{f}, \beta, 0) = minConErr(\vec{f}, \beta, 0)$ will always hold. Hence, the algorithm can be seen as computing either of these values. Intuitively, the algorithm dynamically iterates over all possible divisions of the input into β buckets and returns the minimal sum-square error found.

NoDeletionSummary utilizes two matrices:

- the precomputed matrix *SSE* discussed earlier;
- a matrix M of dimensions $\text{size}(\vec{f}) \times \beta$, in which we will update $M[p, \beta']$ to contain $minArbErr(\vec{f}_{[\text{first}(\vec{f}), p]}, \beta', 0)$.

Hence, $M[\text{last}(\vec{f}), \beta]$ is precisely what we are computing.

The algorithm starts by initializing the matrix M with the error elements for the case of a single bucket (Lines 1–2). Note that this loop, (as well as all other loops in this paper that iterate over elements from $\vec{f}$, such as Line 4 and 6) should be understood as implicitly only looping over elements p for which $\vec{f}[p] > 0$. Thus, for each element p with positive frequency, we update $M[p, 1]$.

To compute the remainder of M, the algorithm uses three nested for loops. In the outer loop, it iterates over the number of β' from 2 to β (Line 3). In the first inner loop, it iterates over elements q from next(first) to last (Line 4). In the inner-most loop, the algorithm iterates over all possible elements p for the left side of the right most bucket (Line 6). Using previously computed values, it then calculates the error, under the assumption that the last bucket ranges from just after p to q, (Line 7) and saves the minimal error in $M[q, \beta']$ (Line 8). Finally, the algorithm returns the minimal error that can be achieved with β buckets, i.e., $M[\text{last}, \beta]$

Algorithm SumThenDelErr($\vec{f}, \beta, k$)

1. $B \leftarrow$ NoDeletionSummaryBuckets($\vec{f}, \beta$)
2. $([p_1, q_1]; n_1) \leftarrow B[1]$
3. **for** $k' \leftarrow 0$ to k
4. **do** $M[1, k'] \leftarrow$ SingleBErr($\vec{f}_{[p_1,q_1]}, k'$)
5. **for** $\beta' \leftarrow 2$ to β
6. **do for** $k' \leftarrow 0$ to k
7. **do** SDUpdateMatrix($\vec{f}, \beta', k', M, B$)
8. **return** $M[\beta, k]$

Algorithm SDUpdateMatrix($\vec{f}, \beta, k, M, B$)

1. $M[\beta, k] \leftarrow \infty$
2. $([p, q]; n) \leftarrow B[\beta]$
3. **for** $k' \leftarrow 0$ to k
4. $e \leftarrow M[\beta - 1, k - k']$
5. $e' \leftarrow$ SingleBErr($\vec{f}_{[p,q]}, k'$)
6. $M[\beta, k] \leftarrow \min\{M[\beta, k], e + e'\}$

Figure 4: Returns either $minArbSDErr(\vec{f}, \beta, k)$ **or** $minConSDErr(\vec{f}, \beta, k)$**, depending on the version of** SingleBucketError **used.**

(Line 9). It is not difficult to see that NoDeletionSummary runs in time $O(\beta\,\text{size}^2)$.

Example 5.1. As a running example, to compare algorithms returning multi-bucket summaries, we consider a dataset with the frequency vector $\vec{f}$, defined as

$$\{1{:}2, 2{:}1, 3{:}2, 4{:}1, 5{:}2, 6{:}3, 7{:}2, 8{:}1\}.$$

The optimal 2-bucket summary of $\vec{f}$, without deletions is

$$B = \{([1, 7]; 13), ([8, 8]; 1)\}$$

with error 2.86. □

Summarize, then Delete. One approach to finding a summary with deletions is to

- *first* compute an optimal summary B without deletions (using NoDeletionSummary) and
- *then* choose the best elements to delete from the buckets, adjusting bucket boundaries accordingly.

We denote the minimal possible error that can be achieved when following this two-step methodology for a dataset with frequency vector $\vec{f}$, bucket bound β and a number k of deletions as $minArbSDErr(\vec{f}, \beta, k)$ and $minConSDErr(\vec{f}, \beta, k)$ for arbitrary and consistent summaries, respectively.

While a-priori it may not be clear how to choose elements to delete, in fact, an optimal choice can be found in polynomial time using dynamic programming. SumThenDelErr in Figure 4 computes the minimal error by filling in a matrix M of dimensions $\beta \times k$. The position $M[\beta', k']$ is updated to be the optimal error of the first β' buckets while allowing up to k' deletions. Hence, the value of interest will be in $M[\beta, k]$ at the end of the algorithm.

Algorithm SumThenDelErr begins by calling an algorithm named NoDeletionSummaryBuckets, which is a version of the previously presented NoDeletionSummary that returns the set of buckets B, for which the error, with no deletions, is minimal. Next the algorithm initializes all entries of M for $\beta = 1$ by calling an algorithm that computes the error for a single bucket (Lines 2–3). In practice, SingleBErr should be replaced with ArbSingleBErr if the goal is to compute $minArbSDErr(\vec{f}, \beta, k)$ and with ConSingleBErr if the goal is to compute $minConSDErr(\vec{f}, \beta, k)$.

Finally, the algorithm considers all choices for $2 \le \beta' \le \beta$, as well as all values of $k' \le k$, by calling SDUpdateMatrix for each combination. Algorithm SDUpdateMatrix is used to update $M[\beta, k]$, by considering all choices k' of how to split the deletions between the first $\beta - 1$ buckets (i.e., $M[\beta - 1, k - k']$) and the final bucket (i.e., SingleBErr($\vec{f}_{[p,q]}, k'$)).

Example 5.2. Recall the frequency vector $\vec{f}$ from Example 5.1. When running SumThenDelErr($\vec{f}$, 2, 2) once with ArbSingleBErr and once with ConSingleBErr we derive the summaries B_1 and B_2, respectively:

$$B_1 = \{([1, 7]; 11), ([8, 8]; 1)\} \qquad err{:}\ 0.18$$
$$B_2 = \{([2, 7]; 11), ([8, 8]; 1)\} \qquad err{:}\ 0.35$$

where the former is derived by deleting two occurrences of element 6, and the latter is derived by removing two occurrences of element 1. □

If we can show that the algorithm returns a value close to $minArbErr(\vec{f}, \beta, k)$ or $minConErr(\vec{f}, \beta, k)$, we can leverage optimizations presented in the past for finding optimal summaries without deletions to solve the problem at hand. Unfortunately, the following theorem states that SumThenDelErr can be arbitrarily bad.

Theorem 5.3. *For any β, k and $e > 0$, there exists a multiset P with frequency vector $\vec{f}$ such that*

$$minArbSDErr(\vec{f}, \beta, k) - minArbErr(\vec{f}, \beta, k) > e\,,$$
$$minConSDErr(\vec{f}, \beta, k) - minConErr(\vec{f}, \beta, k) > e\,.$$

Proof. Due to space limitations, we only show the result for $k = 1$ and $\beta = 2$. The more general case, however, can be shown similarly. Let $e > 0$ be an error gap. Let n be a natural number such that $n > max\{10, 5e\}$. Let $\vec{f}$ be the frequency vector $\{0{:}n, 2{:}1, 4{:}n, 7{:}n\}$.

If we can delete elements before choosing the buckets, the optimal strategy is to reduce the frequency of the element 2 in $\vec{f}$, thus deriving the frequency vector $\vec{f}' = \{0{:}n, 4{:}n, 7{:}n\}$. Then the optimal summary would be $\{([0, 0]; n), ([4, 7]; 2n)\}$. This summary is also consistent. Hence, $minArbErr(\vec{f}, 2, 1) = minConErr(\vec{f}, 2, 1) = n^2$.

There are three possible no-deletion summaries of $\vec{f}$ with two buckets. These are specified below with their errors.

$$B_1{:}\ \{([0, 4]; 2n + 1), ([7, 7]; n)\} \qquad err_1{:}\ \frac{6}{5}n^2 + \frac{4}{5}n + \frac{4}{5}$$
$$B_2{:}\ \{([0, 2]; n + 1), ([4, 7]; 2n)\} \qquad err_2{:}\ \frac{5}{3}n^2 + \frac{2}{3}n + \frac{2}{3}$$
$$B_3{:}\ \{([0, 0]; n), ([2, 7]; 2n + 1)\} \qquad err_3{:}\ \frac{4}{3}n^2 - \frac{2}{3}n + \frac{5}{6}$$

It is not difficult to see that for a sufficiently large n, the optimal summary without deletions is $B_1 = \{([0, 4]; 2n + 1), ([7, 7]; n)\}$. Thus, this summary would then be returned by the algorithm NoDeletionSummaryBuckets.

No deletion can be made from B_1 if the summary must be consistent. Hence, SumThenDelErr would return precisely err_1 if ConSingleBucketError is used. It easily follows that

$$minConSDErr(\vec{f}, \beta, k) - minConErr(\vec{f}, \beta, k) > e.$$

For arbitrary summaries, that need not be consistent, the largest decrement in the error of B_1 that can be achieved by deleting one element is derived by reducing the frequency of element 0

(or equivalently of element 4) by one, which gives us the frequency vector $\vec{f}_1' = \{0{:}n-1, 2{:}1, 4{:}n, 7{:}n\}$ and the summary $B_1' = \{([0,4];2n), ([7,7];n)\}$. Thus, we have that $minArbSDErr(\vec{f}, 2, 1) = \frac{6}{5}n^2 - 2n + 2$.

Now, $minArbSDErr(\vec{f}, 2, 1) - minArbErr(\vec{f}, 2, 1)$ equals

$$\frac{6}{5}n^2 - 2n + 2 - n^2 = \frac{1}{5}n^2 - 2n + 2$$

which is greater than e when n is chosen to be greater than $max\{10, 5e\}$, as required. □

The above theorem implies that one cannot create an optimal summary without deletions, and only afterwards delete elements from the pre-chosen buckets without paying an arbitrarily large penalty in error. However, this algorithm was worth discussion as in practice we will show that it often works quite well. See Section 7 for experimental results.

Delete, then Summarize. An alternative approach that also uses previous work on optimal summaries without deletions is to

- *first* choose k outliers in the dataset to delete, thereby deriving a k-deletion P' of the original dataset P
- *then* compute an optimal summary without deletions for P' (using NoDeletionSummary).

This approach may not return consistent summaries, but it can be used to produce arbitrary summaries. It is difficult to formally prove the quality of the results that this approach will derive. In particular, there are many different ways to define the notion of an outlier. Hence in the first step it is not even clear what we are attempting to delete. Second, most outlier detection algorithms return a set of outliers given a dataset, but cannot be tuned to find a specific number of outliers. This poses a significant technical challenge. Thus, instead of trying to formally prove that this approach will not yield optimal outcomes, we defer consideration of this approach to the experimentation. There we experimentally show that the results returned using this methodology are significantly inferior to all other approaches considered in this paper.

5.2 Optimal Multi-Bucket Summaries

We consider the problem of computing $minArbErr(\vec{f}, \beta, k)$ for arbitrary values of β. Algorithm ArbMultiBErr in Figure 5 is inspired by NoDeletionSummary. ArbMultiBErr uses dynamic programming to fill in a matrix M of dimension size $\times\ \beta \times k$. The position $M[p, \beta', k']$ is updated to be the minimal error of a k'-deletion summary of $\vec{f}_{[\mathsf{first},p]}$ with β' buckets, i.e., $minArbErr(\vec{f}_{[\mathsf{first},p]}, \beta', k')$. Hence, the value of interest will be in $M[\mathsf{last}, \beta, k]$ at the end of the algorithm.

ArbMultiBErr begins by initializing all entries of M for $\beta = 1$ by calling ArbSingleBErr (Lines 1–3). Next, the algorithm considers all choices for $2 \le \beta' \le \beta$, as well as all elements p, other than the first, and all values of $k' \le k$. Algorithm ArbUpdateMatrix is used to update $M[p, \beta', k']$.

Given p, β, k, ArbUpdateMatrix computes $M[p, \beta, k]$, by considering the minimal choice among several options:

- If the frequency value $\vec{f}[p]$ of p is at most k, then we can create β buckets for $\vec{f}_{[\mathsf{first},p]}$ by removing all occurrences of p, and then finding the optimal buckets for $\vec{f}_{[\mathsf{first},\mathsf{prev}(p)]}$ with $k - \vec{f}[p]$ deletions, using the precomputed value in M. (Lines 1–3)

Algorithm ArbMultiBErr($\vec{f}, \beta, k$)
1. **for** $p \leftarrow$ first to last
2. **do for** $k' \leftarrow 0$ to k
3. **do** $M[p, 1, k'] \leftarrow$ ArbSingleBErr($\vec{f}_{[\mathsf{first},p]}, k'$)
4. **for** $\beta' \leftarrow 2$ to β
5. **do for** $p \leftarrow$ next(first) to last
6. **do for** $k' \leftarrow 0$ to k
7. **do** ArbUpdateMatrix($\vec{f}, \beta', k', p, M$)
8. **return** $M[\mathsf{last}, \beta, k]$

Algorithm ArbUpdateMatrix($\vec{f}, \beta, k, p, M$)
1. **if** $\vec{f}[p] \le k$
2. **then** $M[p, \beta, k] \leftarrow M[\mathsf{prev}(p), \beta, k - \vec{f}[p]]$
3. **else** $M[p, \beta, k] \leftarrow \infty$
4. **for** $q \leftarrow$ first to prev(p)
5. **do for** $k' \leftarrow 0$ to k
6. **do** $e \leftarrow M[q, \beta - 1, k - k']$
7. $e' \leftarrow$ LowerMaxError($\vec{f}_{[\mathsf{next}(q),p]}, k'$)
8. $M[p, \beta', k'] \leftarrow \min\{M[p, \beta, k], e + e'\}$

Figure 5: Returns $minArbErr(\vec{f}, \beta, k)$.

- In addition, for every element q preceding p, and for every $k' \le k$ we consider the case that $\beta - 1$ buckets are used to cover $\vec{f}_{[\mathsf{first},q]}$ with $k - k'$ deletions and a final bucket covers the range from next(q) to p, using k' deletions. The error of such a summary is $M[q, \beta - 1, k - k'] +$ LowerMaxError($\vec{f}_{[\mathsf{next}(q),p]}, k'$). (Lines 4–8)

Example 5.4. Recall the frequency vector $\vec{f}$ from Example 5.1. With ArbMultiBErr we will get an error of 0.125 corresponding to the deletion of one occurrence of the element of 6, as well as the element 8, and the buckets $\{([1,4];6), ([5,7];6)\}$. □

Theorem 5.5. *Given $\vec{f}$, β and k,* ArbMultiBErr($\vec{f}, \beta, k$) *computes $minArbErr(\vec{f}, \beta, k)$ in time*

$$O(\beta k^2 \mathsf{size}^2 + \mathsf{size}^2(\mathsf{size}\log\mathsf{size} + k)).$$

Proof. Correctness is easy, and follows from Theorem 4.1, and from the fact that the algorithm considers all possible ways to divide the frequency vector into buckets, and all possible choices of deletions. Note that the algorithm directly considers removals of elements only on the right-hand of buckets. (For the first bucket both sides are considered). However, any removals of elements at the left-hand side of these buckets can equivalently be thought of as removals from the right-hand side of the previous bucket.

To achieve the stated runtime, we pre-compute the values of LowerMaxError($\vec{f}_{[p,q]}, k'$) for all $p < q$ (with non-zero frequencies) and for all $k' \le k$. For a choice of p and q, we can compute LowerMaxError($\vec{f}_{[p,q]}, k'$) for all values of k' in time size log size + k, by sorting $\vec{f}_{[p,q]}$ by frequency values, and then computing the results incrementally starting with $k = 0$. This give a total time of $\mathsf{size}^2(\mathsf{size}\log\mathsf{size} + k)$ for LowerMaxError computations. These values are then read from a pre-computed data structure, in $O(1)$ when running ArbUpdateMatrix.

The remainder of the algorithm runs in time $O(\beta k^2 \mathsf{size}^2)$ due to the nested loops of length β (Line 4 of ArbMultiBErr), of length size (Line 5), of length k (Line 6) and the loop of lengths size (Line 4 of ArbUpdateMatrix) and length k (Line 5). □

Algorithm ConUpdateMatrix($\vec{f}, \beta, k, p, M$)
1. **if** $\vec{f}[p] \leq k$
2. **then** $M[p, \beta, k] \leftarrow M[\text{prev}(p), \beta, k - \vec{f}[p]]$
3. **else** $M[p, \beta, k] \leftarrow \infty$
4. **for** $q \leftarrow \text{prev}(p)$ to first
5. **do** $e \leftarrow M[q, \beta - 1, k] + SSE[\text{next}(q), p]$
6. $M[p, \beta, k] \leftarrow \min\{M[p, \beta, k], e\}$

Figure 6: Used as a sub-procedure in order to compute $minConErr(\vec{f}, \beta, k)$.

5.3 Optimal Consistent Multi-Bucket Summaries

We now consider the problem of computing the error for optimal consistent summaries, i.e., $minConErr(\vec{f}, k, \beta)$. Let ConMultiBErr be an algorithm identical to ArbMultiBErr, except that in Line 3 it calls ConSingleBErr and in Line 7 it calls ConUpdateMatrix. Recall that ConSingleBErr appears in Figure 2. ConUpdateMatrix appears in Figure 6. It remains to explain how this algorithm works.

ConUpdateMatrix($\vec{f}, \beta, k, b, M$) is called to update the value in $M[p, \beta, k]$ for $\beta > 1$. It first considers the case that the right-most element (which is now p) can be completely removed (Lines 1–3), in which case the previously computed error can be used. It then (Lines 4–6) considers all splits of $\vec{f}$ into two, such that $\beta - 1$ buckets are used to summarize $\vec{f}_{[\text{first},q]}$ (with error appearing in $M[q, \beta - 1, k]$) and one bucket is used to summarize $\vec{f}_{[\text{next}(q),p]}$ (with error appearing in $SSE[\text{next}(q), p]$). Note that in this case, we can assume that no deletions occur in the rightmost bucket, as:

(1) we already considered the case that occurrences of p are removed and
(2) any removals of elements at the left-hand side of the final bucket can equivalently be thought of as removals from the right-hand side of the previous bucket.

Example 5.6. Recall once again $\vec{f}$ from Example 5.1. With algorithm ConMultiBErr we will get an error of 0.17 corresponding to the deletion of the element of 4, as well as the element 8, and the buckets $\{([1, 3]; 5), ([5, 7]; 7)\}$. □

We can show the following result. Observe that ConMultiBError is only slower than algorithm NoDeletionSummary by a factor of k.

Theorem 5.7. *For a given $\vec{f}$, β and k,* ConMultiBErr($\vec{f}, \beta, k$) *computes $minConErr(\vec{f}, \beta, k)$ in time $O(\beta\, k\, \text{size}^2)$.*

Proof. Correctness is easy to show as the different ways to split $\vec{f}$ to different buckets, while deleting all occurrences of elements not in these buckets, is considered. To show the runtime, observe that ConSingleBErr runs in $O(\text{size})$. Therefore, Lines 1–3 of ConMultiBErr run in time $O(k\, \text{size}^2)$. ConUpdateMatrix runs in time $O(\text{size})$. Therefore, Lines 4–7 of ConMultiBErr run in time $O(\beta\, k\, \text{size}^2)$, as required. □

6 SUMMARIES OF DATASETS WITH MULTIPLE COLUMNS

We now briefly consider the problem of finding an optimal summary for datasets with multiple columns, in the presence of outliers. Let $\mathcal{R}$ be a relation with n columns, $(\beta_1, \ldots, \beta_n)$ be a n- tuple indicating the number of buckets per column that can be allocated and let k be a number. We consider two types of outliers:

- *value deletions*: a specific value in some tuple t can be considered an outlier in a specific column (even though the other values in t are not outliers), and thus, be deleted from the column;
- *tuple deletions*: a tuple t can be considered an outlier, and thus, be deleted from the relation.

Our goal is to find a summary with β_i buckets for column i, that minimizes the sum of error over all columns, when there can be at most k value or tuple deletions.

	C	X
t_1^1	2	10
t_2^1	2	20
t_3^1	2	30
t_1^2	4	20
t_2^2	4	30
t_3^2	4	40
t_1^3	6	10
t_2^3	6	40
t_3^3	6	50
t^0	1	0
t^1	3	0
t^2	5	0
t^3	7	0

$\mathcal{R}$

	C	X
t_1^1	2	10
t_2^1	2	20
t_3^1	2	30
t_1^2	4	20
t_2^2	4	30
t_3^2	4	40
t_1^3	6	10
t_2^3	6	40
t_3^3	6	50
t^0	1	0
t^1	3	0
t^2	5	0
t^3	7	0

$\mathcal{R}$

Figure 7: Example multicolumn dataset

Example 6.1. Consider the relation $\mathcal{R}$ repeated twice in Figure 7. (Ignore the first column which names the tuples for convenience.) Suppose $k = 6$. In the version of $\mathcal{R}$ on the left, we have chosen up to k values to be deleted per column, while in the version on the right we have chosen up to k tuples to be deleted. (Both are indicated by the color blue.)

Now, suppose we can allocate one bucket for the first column C and three buckets for the second column X. For both types of deletions we can find a *perfect summary* (i.e., one with no error). For the deletions of the left we would choose buckets:

- $([1, 7]; 7)$ for column C and
- $([0, 0]; 4)$, $([20, 20]; 2)$, $([30, 30]; 2)$ for column X.

For the deletions on the right we would choose buckets:

- $([1, 7]; 7)$ for column C and
- $([0, 0]; 4)$, $([20, 20]; 1)$, $([40, 40]; 2)$ for column X.

The above example demonstrated summaries with $(1, 3)$ buckets for the columns and six value/tuples deletions that are clearly optimal (as they have no error at all). The problem arises as to how hard it is to find such optimal summaries in the general case.

Theorem 6.2. *Let $\mathcal{R}$ be a relation with n columns, let $(\beta_1, \ldots, \beta_n)$ be the bucket size bounds and let k be natural number. Then:*

(1) *An optimal summary satisfying the bucket size bounds, with up to k value deletions per column can be found in polynomial time.*
(2) *The problem of determining whether there exists a perfect summary of $\mathcal{R}$ with up to k tuple deletions is NP-complete, even if*
   - *the dataset has two attributes* or
   - *all attributes can have at most two buckets.*

The first claim is easy to show as we can simply apply the algorithms from the previous section to each column separately. For the second claim, in the case in which there are only two attributes we can show hardness by a reduction from the monotone satisfying assignment problem. For the case where there are at most two buckets per attribute we can show hardness by a reduction from 1-in-3 SAT. The proofs are omitted due to space limitations, but the main ideas of the second claim are demonstrated in the following examples.

Example 6.3. We demonstrate hardness when the relation contains only two columns.

Let ψ be a positive 3-SAT formula with m clauses $C_1, \ldots, C_m$, over the variables $x_1, \ldots, x_n$. We say that ψ is h-monotone satisfiable if ψ can be satisfied by an assignment in which at most h variables are assigned true. This is a well-known NP-complete problem.

Let $\mathcal{R}$ be a relation of cardinality 2, containing two types of tuples:

- For each clause $C_i = x_{i_1} \vee x_{i_2} \vee x_{i_3}$ we add to $\mathcal{D}$ three tuples $t_1^i = (2i, 10i_1)$, $t_2^i = (2i, 10i_2)$, $t_3^i = (2i, 10i_3)$.
- For every $0 \geq j \geq m$ we add the tuple $t^j = (2j + 1, 0)$.

Observe that $\mathcal{R}$ contains $4m + 1$ tuples. For example, the formula

$$(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_4 \vee x_5)$$

corresponds to the relation $\mathcal{R}$ in Figure 7.

Now, we choose $k = 2m$ and $(1, h + 1)$ as our bucket bound, i.e., a single bucket for the first column and $h + 1$ buckets for the second column. We can show that ψ is h-monotone if and only if there is a k-tuple deletion for which there is a summary bounded by $(1, h + 1)$ with zero error.

For example, suppose that $h = 2$. Consider the k-tuple deletion $\mathcal{R}'$ derived by removing the blue tuples in the relation on the right. It is easy to see that we can use a single bucket for the first column, and $h + 1 = 3$ buckets for the second column, to derive a summary with error of zero. Observe also that $\mathcal{R}'$ corresponds to a satisfying assignment that assigns at most h variables the value true (in this case x_2 and x_4 as the elements 20 and 40 remain after the k-tuple deletion).

One can show (omitting some precise details due to space limitations) that the opposite holds too, i.e., that any k-deletion that has a summary bounded by $(1, h + 1)$ with an error of zero corresponds to an h-monotone satisfying assignment. □

Example 6.4. We demonstrate why determining whether there exists a k-tuple deletion $\mathcal{R}'$ with a perfect summary S is a difficult problem, even when there are only two buckets per column.

Consider a positive 3-SAT formula with m clauses $C_1, \ldots, C_m$, over the variables $x_1, \ldots, x_n$. Recall that the 1-in-3 SAT problem is to determine whether there exists an assignment that satisfies precisely one variable in each clause.

We create a relation $\mathcal{R}$ with $n + 1$ columns, and three types of tuples:

	X_1	X_2	X_3	X_4	X_5	C
t_1^1	2	-2	-2	0	0	1
t_2^1	-2	2	-2	0	0	1
t_3^1	-2	-2	2	0	0	1
t_1^2	0	2	-2	-2	0	2
t_2^2	0	-2	-2	-2	0	2
t_3^2	0	-2	-2	2	0	2
t_1^3	2	0	0	-2	-2	3
t_2^3	-2	0	0	2	-2	3
t_3^3	-2	0	0	-2	2	3
$t_1^{c,1}$	0	0	0	0	0	1
$t_2^{c,1}$	0	0	0	0	0	1
$t_1^{c,2}$	0	0	0	0	0	2
$t_2^{c,2}$	0	0	0	0	0	2
$t_1^{c,3}$	0	0	0	0	0	3
$t_2^{c,3}$	0	0	0	0	0	3
t_1^0	0	0	0	0	0	0
t_2^0	0	0	0	0	0	0
t_3^0	0	0	0	0	0	0

Figure 8: Example relation in 1-in-3 SAT reduction.

- For each clause $C_i = x_{i_1} \vee x_{i_2} \vee x_{i_3}$ we add three tuples as follows:
  - for all $j = 1, 2, 3$ we have $t_j^i[n + 1] = i$;
  - for all $j = 1, 2, 3$ and $k \notin \{i_1, i_2, i_3\}$ we have $t_j^i[k] = 0$;
  - for all $j = 1, 2, 3$ and $k = 1, 2, 3$ we have $t_j^i[i_k] = 2$ if $k = j$ and $t_j^i[i_k] = -2$, otherwise.
- For each $i \leq m$, we add two tuples $t_1^{c,i} = t_2^{c,i} = (0, \ldots, 0, i)$.
- Finally, we add three tuples $t_1^0 = t_2^0 = t_3^0 = (0, \ldots, 0)$.

In total, $\mathcal{R}$ has $5m + 3$ tuples. For example, the formula

$$(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_4 \vee x_5)$$

corresponds to the dataset in Figure 8.

Now, suppose that $k = 2m$ and our bucket bound allows two buckets per attribute. For the example in Figure 8, we can remove up to 6 tuples. It is not difficult to see that if we remove the blue tuples, we are left with a data set $\mathcal{R}'$ for which we can create a summary with error of zero. This holds since each column $x_1, \ldots, x_5$ has precisely two different elements (which can be placed in different buckets) and column C has three occurrences each of every value from 0 to 3. Observe also that the remaining rows define an assignment that satisfies precisely one variable in each clause, if we assign each x_i true if $\mathcal{R}'[X_i]$ contains element 2, and false otherwise. (In the example given this corresponds to mapping x_2 and x_5 to true.)

Indeed, with a bit of reasoning it is possible to show the opposite direction too, i.e., a k-tuple deletion $\mathcal{R}'$ for which there is a summary with error of zero corresponds to a solution to the 1-in-3 SAT problem. Intuitively, this holds since such a tuple-deletion must retain, for each x_i, only the elements 0 and 2 or only the elements 0 and -2 (thus, corresponding to a truth assignment).

Dataset	size_P	$\text{size}_{\vec{f}}$
ForestAspect	581,012	361
ForestDistHydro	581,012	136
IncomeCapitalGain	48,842	123
IncomeHoursWeek	48,842	96

Figure 9: Real datasets.

In addition, we must retain precisely one tuple from every triple t_1^i, t_2^i, t_3^i so as to have error of zero on the column C, i.e., we will map precisely one variable in each clause to the value true.[3] □

7 EXPERIMENTAL RESULTS

In the following we discuss our experimental results which consider different datasets, and varying β and percentage of elements that can be deleted, denoted ρ. As default values, we use $\beta = 10$ and $\rho = 2\%$. In our experiments, we prefer to consider the percentage of elements to be deleted (ρ) instead of an absolute number (k), as the size of the dataset should determine the number of elements we are willing to delete. We set a time limit of five minutes, i.e., tests that exceeded the time limit were excluded from the results.

All experimentation was run on a standard Win7 desktop with 16GB RAM and an Intel i5-4570 processor. The algorithms were implemented in Java, and experimentation was run with a limit of 1GB of main memory.

Algorithms. We implemented five algorithms for the k-deletion problem. We abbreviate the algorithm names as follows. We use Arb for ArbMultiBErr and Con for ConMultiBErr. We implemented algorithm SumThenDelErr with ArbSingleBErr, and ConSingleBErr, henceforth referred to as SDArb and SDCon, respectively. Finally, we implemented the *delete then summarize* strategy using a well-known distance-based outlier detection algorithm [4] to remove outliers and then summarizing the result using NoDeletionSummary. This algorithm is referred to as DS.

Datasets. We run our algorithms on both synthetic datasets and real datasets. The synthetic datasets were generated by two different distributions—randomly permuted zipf distribution with skew parameter $z = 0.85$ (as in [11]), and normal distribution with variance proportional to 25% of the value range. For each experiment over the synthetic datasets, we generated three instances of the dataset with the same parameters, ran the tests on all three, and computed the average result. Our default synthetic datasets have 50,000 elements and a range of 100, i.e., the elements were sampled from $[1, 100]$.

We also use several real datasets, taken from the UCI KDD Archive[4] and the UCI Machine learning repository[5]. When considering the size of these datasets, there are two factors of importance: the number of elements in the dataset, called size_P and the number of different elements in the dataset (i.e., the number of non-zero elements in the frequency vector), called $\text{size}_{\vec{f}}$. Note that $\text{size}_{\vec{f}}$ corresponds to the value size discussed in the previous sections. The dataset sizes are summarized in Figure 9.

[3]More details are required to make a precise proof, e.g., an assumption that no variable appears in all clauses, but these are omitted due to space limitations.
[4]http://kdd.ics.uci.edu/
[5]http://archive.ics.uci.edu/ml/index.php

The Forest dataset contains data obtained from US Forest Service, where the Aspect column is aspect in degrees azimuth, and DistHydro is the horizontal distance to nearest surface water features. For DistHydro we rounded to the nearest 10 meters. The Income dataset (referred to as Adult in the repository) was originally extracted from the Census Bureau 1994 database. We use the columns CapitalGain and HoursWeek, which is the number of work hours per week.

In all runtime graphs, the y-axis is in log scale and the units are in seconds. In all error graphs, the data points represent the ratio of the error in the current setting to the minimum error of the same settings with no deletions. Intuitively, lower values indicate a larger percentage of reduction in error due to deletions.

Comparing All algorithms. In our first test, we compare the error and runtime of all algorithms over the Income dataset and synthetic dataset, using the default values for β and ρ. The results of this test appear in Figures 10a, 10b, 10c and 10d. Due to space limitations, we omit the Forest dataset from this experiment, but similar trends can be seen there.

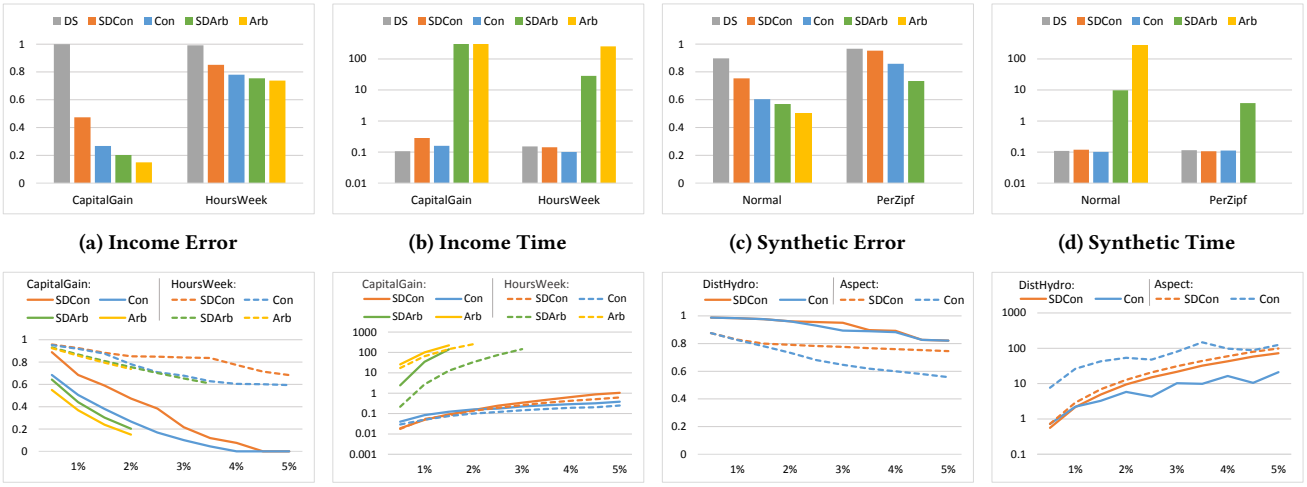
Algorithm DS, which uses distance based outlier detection, is mostly unsuccessful in improving the error. In our testing, there have even been cases in which the error increases using this algorithm. This is not surprising, both due to the fact that outliers are removed before determining bucket boundaries and because the algorithm does not always even produce enough outliers to delete. This result was consistent in all experiments and therefore, we do not consider it in remainder of the experimentation.

For the other algorithms, the graphs show a consistent order of the error reduction—Arb always finds the optimal error, then SDArb, Con and finally SDCon have increasing errors. The runtime performance, on the other hand, is in almost the opposite order, albeit Con and SDCon are very close in runtime. One can observe that Con achieves error that is fairly close to Arb and SDArb with runtime that is better in orders of magnitude.

Note that the runtime of Arb is so poor over the permuted zipf dataset, that it timed out, and therefore its runtime and error are missing. This occurs often in the experiments both for Arb and for SDArb, as these algorithms have a runtime that is a function of k^2, even when a single bucket is used.

Varying percentage of Deletions. We tested the effect of changing the percentage of elements deleted by varying the value of ρ, and running our algorithms with k chosen as $\rho \cdot \text{size}_P$. The error and runtime results of these tests appear in Figures 10e, 10g, 11a ,10f, 10h and 11b. Note that for the synthetic datasets, we present the error for both permuted zipf and normal distribution, but the runtime only for the former. Since the runtime is not affected by the data distribution, the graph omitted is almost identical to the one appearing.

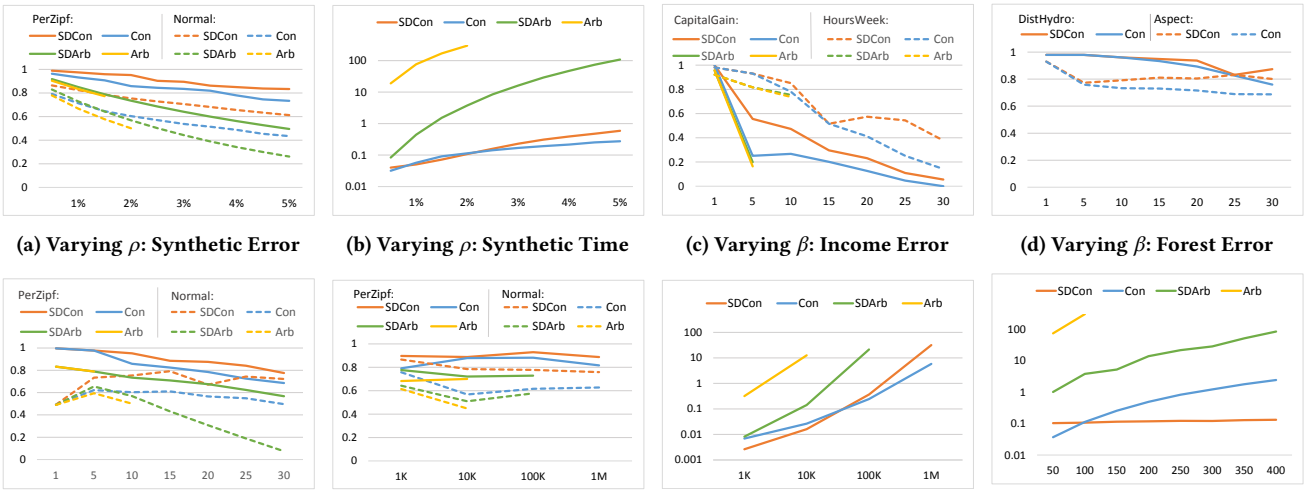
As expected, in all algorithms and datasets, the error is reduced as ρ grows. The degree to which the error is reduced differs between the datasets, due to differences in the data distributions. For the synthetic data, deletions are more significant in normal distribution than in permuted zipf distribution. Almost consistently, Con and SDCon have significantly lower runtimes than SDArb which is much faster than Arb. It is interesting to see that in most tests, on lower values of ρ, SDCon is faster than Con, but as ρ increases, SDCon becomes slower. The only exception in our tests was on the dataset ForestAspect, where the number of distinct values i.e., $\text{size}_{\vec{f}}$, is much higher than in the other



(a) Income Error (b) Income Time (c) Synthetic Error (d) Synthetic Time

(e) Varying ρ: Income Error (f) Varying ρ: Income Time (g) Varying ρ: Forest Error (h) Varying ρ: Forest Time

Figure 10: Experiments for default values and varying ρ over real datasets.



(a) Varying ρ: Synthetic Error (b) Varying ρ: Synthetic Time (c) Varying β: Income Error (d) Varying β: Forest Error

(e) Varying β: Synthetic Error (f) Varying size_P: Error (g) Varying size_P: Time (h) Varying $\text{size}_{\vec{f}}$: Time

Figure 11: Experiments for varying ρ, β, size_P and $\text{size}_{\vec{f}}$

datasets, and larger $\text{size}_{\vec{f}}$ degrades the runtime of Con. This phenomenon will be discussed later in our experiments, when we consider varying the value $\text{size}_{\vec{f}}$.

Varying Number of Buckets. Next, we tested the results of our algorithms when varying the number of buckets. The runtimes, which do not appear in graphs due to space limitations, increase linearly together with β. The error reduction is shown in Figures 11c, 11d, and 11e. In general, as the number of buckets increase, the error decreases. On IncomeCapitalGain for instance, the error goes down to zero with 30 buckets, but On ForestDistHydro on the other hand, the improvement in error with 30 bucket is only a little under 80%. There are also cases where the reduction in error actually increases when buckets are added. This counter-intuitive result is because the error ratio is computed with respect to the *same settings* (and thus, the same number of buckets) without deletions. As the number of buckets increases, the relative gain by removing elements may decrease. This is particularly noticeable for SDCon.

Varying the Values of size_P *and* $\text{size}_{\vec{f}}$. We studied how the total number of elements size_P affects the results. To this end, we created synthetic datasets of increasing size. In Figures 11f and 11g depict the change in error and in runtime over different sizes of multisets of elements. The relative error remains approximately the same even as the dataset size increases. This apparently is the result of the fact that the proportion of elements deleted relative to the entire set remains the same. In contrast, the runtime grows linearly with the size of the dataset. (Note that both axes of Figure 11g are on log scale.) The linear increase in runtime occurs as the constant 2% element deletion translates to bigger absolute numbers of k-deletions as the dataset size increases. Similarly to before, for larger datasets (i.e., larger values of k), SDCon runs slower than Con.

In the next experiment we considered increasing the number of distinct elements $\text{size}_{\vec{f}}$. In this experiment the dataset remained at the default size of 50,000 elements, but we increased the range of values so as to increase $\text{size}_{\vec{f}}$. Due to space limitations, the

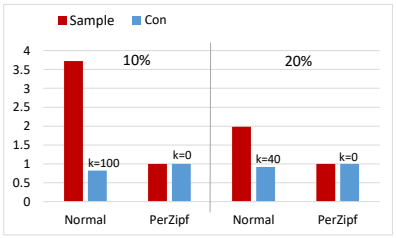


Figure 12: Error of Sampling versus Optimal Deletion

error graph is omitted, as it had the same trends as in Figure 11f. Figure 11h shows the runtime as size $\vec{f}$ increases. As expected, all algorithms have runtime that grow with size $\vec{f}$, except for SDCon whose runtime is independent of size $\vec{f}$.

Comparison with Sampling for Summary Construction. Since histogram construction is costly, in many systems histograms are computed over a sample of the dataset. Intuitively, it might appear that such a sampling based approach will yield similar results to our algorithms that allow for optimal deletions, as perhaps outliers will not be chosen in the sample. In order to check how sampling affects the error in practice, for each dataset P, we constructed a summary B over sampled data using the NoDeletionSummary algorithm. Some of the elements in the original dataset P did not fall into the range of any of the buckets, and were considered to have been "deleted" from the dataset, yielding a new dataset P'. We then computed $err(B, P')$ and compared this value with the error of the optimal consistent summary allowing for $|P - P'|$ deletions.

The results of this experiment on the synthetic datasets, with sampling of 10% and 20% are depicted in Figure 12. (The results on the real datasets were similar.) The size of $|P - P'|$, i.e., the value that was then used as k in the input of ConMultiBErr, appears on top of the bars. The results show two cases. With the permuted Zipf dataset there were no deletions at all as a result of the sampling, and thus, the error is exactly the same as the original error of NoDeletionSummary algorithm. With the normal dataset, there was a larger number of deletions, but the elements deleted by the sampling actually increased the error significantly. When ConMultiBErr was run with the same number of deletions, the error was reduced. We conclude that sampling cannot be used as an effective technique in order to reduce the error of a summary.

8 CONCLUSION

This paper studied the problem of an optimal summary with β buckets, when k outliers need not be covered. We presented the first algorithms for this problem, by taking two different approaches for deleting elements (arbitrarily and consistently), and attempting to determine which elements to delete at different stages (before, during and after finding the optimal summary). We also considered the problem of multi-column datasets. The experimentation shows that algorithm Con has the best balance between low error and low runtime. In addition, on smaller datasets Arb performs very well, providing significantly lower error.

As future work, we intend to consider domains in which there is no natural (useful) ordering over the values, and hence, buckets cannot be described by their endpoints. While our algorithms work well for moderately large datasets, they degrade when the dataset becomes huge. We intend to develop approximation algorithms to deal with this case. Finally, another important direction is finding optimal summaries with outliers over streaming data.

ACKNOWLEDGMENTS

The authors were partially supported by the Israel Science Foundation (Grant 879/16).

REFERENCES

[1] Hee-Kap Ahn, Sang Won Bae, Erik D. Demaine, Martin L. Demaine, Sang-Sub Kim, Matias Korman, Iris Reinbacher, and Wanbin Son. 2011. Covering Points by Disjoint Boxes with Outliers. *Comput. Geom. Theory Appl.* 44, 3 (April 2011), 178–190. https://doi.org/10.1016/j.comgeo.2010.10.002

[2] Rossen Atanassov, Prosenjit Bose, Mathieu Couture, Anil Maheshwari, Pat Morin, Michel Paquette, Michiel Smid, and Stefanie Wuhrer. 2009. Algorithms for optimal outlier removal. *Journal of discrete algorithms* 7, 2 (2009), 239–248.

[3] Sang Won Bae. 2019. Computing a minimum-width square or rectangular annulus with outliers. *Computational Geometry* 76 (2019), 33 – 45. https://doi.org/10.1016/j.comgeo.2018.08.002

[4] Stephen D. Bay and Mark Schwabacher. 2003. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *SIGKDD*. 29–38. https://doi.org/10.1145/956750.956758

[5] Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. 1998. Random Sampling for Histogram Construction: How much is enough?. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*. 436–447. https://doi.org/10.1145/276304.276343

[6] Matthew Gebski and Raymond K. Wong. 2007. An Efficient Histogram Method for Outlier Detection. In *DASFAA*. 176–187. https://doi.org/10.1007/978-3-540-71703-4_17

[7] S. Guha and N. Koudas. 2002. Approximating a data stream for querying and estimation: algorithms and performance evaluation. In *Proceedings 18th International Conference on Data Engineering*. 567–576. https://doi.org/10.1109/ICDE.2002.994775

[8] Sudipto Guha, Nick Koudas, and Kyuseok Shim. 2001. Data-streams and Histograms. In *STOC*. ACM, New York, NY, USA, 471–475. https://doi.org/10.1145/380752.380841

[9] Yannis E. Ioannidis and Viswanath Poosala. 1995. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. *SIGMOD Rec.* 24, 2 (May 1995), 233–244. https://doi.org/10.1145/568271.223841

[10] H. V. Jagadish, Hui Jin, Beng Chin Ooi, and Kian-Lee Tan. 2001. Global Optimization of Histograms. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*. 223–234. https://doi.org/10.1145/375663.375687

[11] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. 1998. Optimal Histograms with Quality Guarantees. In *VLDB*. 275–286. http://www.vldb.org/conf/1998/p275.pdf

[12] Viswanath Poosala, Yannis E. Ioannidis, Peter J. Haas, and Eugene J. Shekita. 1996. Improved Histograms for Selectivity Estimation of Range Predicates. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*. 294–305. https://doi.org/10.1145/233269.233342

[13] Saket Sathe and Charu C. Aggarwal. 2018. Subspace Histograms for Outlier Detection in Linear Time. *Knowl. Inf. Syst.* 56, 3 (Sept. 2018), 691–715. https://doi.org/10.1007/s10115-017-1148-8

[14] Yizhou Yan, Lei Cao, and Elke A. Rundensteiner. 2017. Scalable Top-n Local Outlier Detection. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. ACM, New York, NY, USA, 1235–1244. https://doi.org/10.1145/3097983.3098191