Improving Accuracy and Robustness of Self-Tuning Histograms by Subspace Clustering

Andranik Khachatryan, Emmanuel Müller, Christian Stier and Klemens Böhm

Abstract—In large databases, the amount and the complexity of the data calls for data summarization techniques. Such summaries are used to assist fast approximate query answering or query optimization. Histograms are a prominent class of model-free data summaries and are widely used in database systems. So-called *self-tuning histograms* look at query-execution results to refine themselves. An assumption with such histograms, which has not been questioned so far, is that they can learn the dataset from scratch, that is – starting with an empty bucket configuration. We show that this is not the case. Self-tuning methods are very sensitive to the initial configuration. Three major problems stem from this. Traditional self-tuning is unable to learn projections of multi-dimensional data, is sensitive to the order of queries, and reaches only local optima with high estimation errors. We show how to improve a self-tuning method significantly by starting with a carefully chosen initial configuration. We propose initialization by dense subspace clusters in projections of the data, which improves both accuracy and robustness of self-tuning. Our experiments on different datasets show that the error rate is typically halved compared to the uninitialized version.

Index Terms—Query Optimization, Selectivity Estimation, Adaptive Histograms, Subspace Clustering.

1 Introduction

Histograms are a fundamental data summarization technique. They are *model-free*, that is – they do not assume any specific distribution of the data. Histograms are used in Query Optimization, which is a core task for any database system. Other applications of histograms are *Approximate Query Processing*, *Spatio-Temporal Queries*, *Top-K Queries*, and *Skylines*. All of these are widely used in commercial and scientific applications. We will use a query optimization scenario throughout this paper. Query optimizers need estimates of query predicate selectivities to accurately estimate the costs of different plans. The majority of commercial and non-commercial DBMS rely on histograms to obtain selectivity estimates. The selectivities are essential when comparing physical access methods, e.g. scan vs index seek, as well as when choosing the method and the order of joins [4]. Therefore, improving histograms a has direct positive impact on query optimization.

When the query predicate refers to more than one attribute, a joint distribution of attributes is needed to obtain the selectivity estimates.

- *A. Khachatryan is with Armsoft LLC, Charents 1, 0025 Yerevan, Armenia. E-mail: andranik@armsoft.am*
- *E. Müller and K. Böhm are with Karslruhe Institute of Technology, 76131 Karlsruhe, Germany. Email: {emmanuel.mueller, klemens.boehm}@kit.edu*
- *C. Stier is with FZI Research Center for Information Technology, Software Engineering, Haid-und-Neu-Straße 10-14, 76131 Karlsruhe Email: stier@fzi.de*

There are two different paradigms of histogram construction: *Static* and *Self-Tuning* histograms.

Static Histograms and Dimensionality Reduction. [5] provides a comprehensive review on histograms. Static histograms [23], [8], [1], [28], [21] are constructed by scanning the entire dataset. They need to be rebuilt regularly to reflect any changes in the dataset. Different notions of histogram optimality have been proposed [10], [12], [5]. However, optimal histogram construction is computationally expensive. Building a static multi-dimensional histogram in full attribute space can also be expensive, both regarding construction time and the space occupied – even when we are not restricting ourselves to optimal histograms. Histograms need to be precise, have small memory footprint and have scalable construction/maintenance times. This is hard to achieve, and traditionally histogram construction has aimed for scalability against number of tuples, sacrificing scalability against number of dimensions [5]. *Dimensionality reduction techniques* try to solve the problem by removing less relevant attributes [7], [6], [11]. Another approach is the SASH [18] framework, which handles memory allocation, refinement and reorganization of histograms. It also decides which attributes to build the histogram on. Skipping of attributes is done for the whole data space, similarly to dimensionality reduction techniques. Overall, SASH is a framework above histograms, as it has to rely on some kind of histogram as underlying data structure (e.g., MHist from [23]).

Dimensionality reduction techniques, however, do not solve the problem in general. Consider a database

relation `Cars( Model, Manufacturer, Year, Color)` [1]. The following correlations are possible:

- `Model` and `Manufacturer`, e.g., Golf implies Volkswagen.
- `Model` and `Year`, e.g., the Volkswagen Beetle was built until 2003.
- `Manufacturer` and `Color`, e.g., Ferraris are typically red.

This example shows that, even for a 3-dimensional dataset, correlations can appear in the projections[2] of attribute-value space. Moreover, correlations can be *local*, e.g., there is strong correlation between `Manufacturer` and `Color` for some values of the attributes, but for rest of the values those attributes are close to being independent. The conventional approach has been to ignore such local correlations and aim for a full-dimensional histogram. This can be wasteful and negatively affect the precision and the memory footprint of the histogram.

Storing the correlation between `Manufacturer` and `Model` does not necessarily mean building a full histogram on these two attributes. If the correlation is strong only for a small sub-region of the two-dimensional attribute-value space, we can store only supplementary statistics for this sub-region. The problem with conventional histogram construction algorithms is that they do not have mechanisms to *detect* local correlations.

It should be stressed that the phenomenon of locally relevant attributes is commonplace. For example, the Sloan Digital Sky Survey (SDSS) dataset [25] contains local correlations, i.e., specific regions of the sky showing high values for specific filters (cf. Section 5). There are techniques which take advantage of such correlations if the system knows about them. An example is the "filtered statistics" in Microsoft SQL Server: one can build a histogram on `Manufacturer` given the filter `Color= 'Red'`. The problem with such techniques is that, to enable them, the user has to know about the correlation. This is practically impossible for large, complex, multi-dimensional data.

Traditional Self-Tuning Histograms. In contrast to static histograms, self-tuning histograms [3], [27], [19], [13] use query feedback to learn the dataset. They amortize the construction costs, because the histogram is constructed on the fly as queries are executed. Self-tuning histograms have been demonstrated to be competitive alternatives to static histograms in terms of estimation accuracy [3], [29], [27], [5]. They are adaptive to query patterns of the user, and stay up-to-date to the data, i.e., unlike static histograms, one does not need to re-build them regularly. As one representative, we consider the data structure of STHoles [3], which is very flexible and has been used in several other histograms [27], [24], [13]. STHoles tries to find rectangular regions in the dataspace which have close to uniform density. However, similarly to traditional index structures such as R-Trees [9], they fail in high dimensional data spaces due to the curse of dimensionality [2] and are affected by the order of tree construction steps [26].

Self-Tuning Histograms and Subspace Clustering. We focus on the general assumption with self-tuning methods, namely that they can learn the dataset from scratch – starting with no buckets and relying only on query feedback. We show that this is not the case. The first few queries define the top-level bucket structure of the histogram; if this structure is bad, then, regardless of the amount of further training, the histogram is unlikely to become good. In general, the self-tuning procedures should be used to *refine* the histogram. When a few good top-level buckets of the histogram are given, self-tuning can be used to successfully "zoom in" into these buckets. However, the converse is not true. Given many buckets with small volume, it is hard to construct an adequate top-level of the tree.

In particular, we observe that this trend strengthens as the dimensionality of the dataset grows. Self-tuning histograms share the same problem with static histograms: they either pick some attributes statically and build the histogram on them, or ignore the issue of locally correlated data by storing full-dimensional buckets. We show that in the later case the histogram is unable to learn important local correlations of data during future training. This, again, can be attributed to a bad initial top-level bucket structure.

To solve this top-level construction problem, we start with an initial configuration which is obtained using subspace clustering [17], [22], [20]. A subspace is an axis-parallel projection (i.e., a projection to a subset of the dimensions) of the data. As a first step [14], we have proposed a method to transfer arbitrary subspace clustering results into an initial histogram structure. We have focused on the clustering aspect and have evaluated the performance of several clustering algorithms. The main result of [14] is that Mineclus [30] performs best among six subspace clustering algorithms as an initializer. Thus, we take it as our basis for this paper. [15] indicates that the order of the learning queries may affect the quality of the histogram, and offers subspace clustering-based initialization. However, our previous work does not investigate the specific reasons *why* non-initialized self-tuning encounters problems, nor does it investigate the mechanisms *how* subspace clustering improves the estimation accuracy. We investigate these reasons and mechanisms here, and highlight the potential of self-tuning when coupled with initialization.

Summing up, we show that self-tuning has the following issues:

1. In the rest of the paper, we focus on the generic case when the domain of the attributes is numeric. Categorical attributes, like `Model` in current example, can be mapped to integers.

2. Projection is any proper subset of the full set of attributes.

- **Sensitivity to learning.** This includes sensitivity to the type, shape, volume and order of the queries.
- **Stagnation.** The histogram reaches only local optima due to missing initial configuration at the top levels of underlying data structures. It does not improve regardless of the amount of further training.
- **Dimensionality.** Inability to learn local correlations, which remain hidden in projections of multi-dimensional datasets.

We demonstrate that these problems exist and formally analyse the reasons why they occur. In particular, we show how the histogram can stagnate by being unable to detect clusters. The reason for stagnation is that a cluster is much harder to detect when one does not know the boundaries of the cluster and has to "assemble" it from smaller parts. We show that such a process (trying to assemble the clusters "bottom-up") can often stall. This is true even if the cluster has a very simple structure.

We propose an initialization method which reduces the error rates of the histogram for large, multi-dimensional datasets. In addition, we show formally that initial buckets can make self-tuning less sensitive to learning. For the initialization we use subspace clustering in order to detect dense clusters in arbitrary projections of high-dimensional data [20], [14]. In a nutshell, these subspace clusters provide essential information for top level buckets and their relevant dimensions.

When conducting this work, we have spent most of our effort to understand *why* the problems described above occur. After that, we propose a solution which addresses them all at once. Initialization by subspace clustering is a natural yet innovative solution to the three problems of self-tuning.

We think that separating the problem analysis from the solution makes our contributions even more clear. First, we analyse in detail the problems with self-tuning: this analysis can be a starting point for others to come up with solutions to problems of self-tuning which would differ from ours. Second, we show that subspace clustering is a promising concept that can be combined with the histogram construction process. Again, this can prompt further research, where ideas from subspace clustering are applied in histogram construction.

2 Self-Tuning Histograms and Their Problems

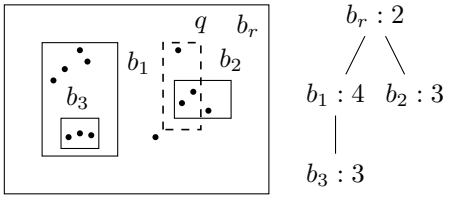
We use STHoles [3] as a representative of self-tuning histograms and describe its main properties and problems. We do this in four steps. First, we describe how the histogram partitions the data space and estimates query cardinalities. Then we describe how new buckets are inserted into the histogram. Third, we describe how the histogram compacts itself by removing buckets to free up space. Last, we derive the open challenges in this processing.

2.1 Histogram Structure and Cardinality Estimation.

STHoles partitions the data space into a tree of rectangular buckets. A bucket b stores the number of tuples in it – $n(b)$. It does not include the tuples in child buckets. Similarly, the volume $vol()$ of a bucket is the volume of the rectangular space occupied by it, without the child buckets. Figure 1 shows a histogram with 4 buckets (solid rectangles) and a query (the dashed rectangle). STHoles estimates the cardinality of query q using the *Uniformity Assumption*. This means that it assumes the tuples inside the buckets to be distributed uniformly:

$$est(q, H) = \sum_{b \in H} n(b) \cdot \frac{vol(q \cap b)}{vol(b)} \quad (1)$$

where H is the histogram.



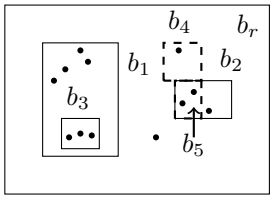
(a) STHoles with 4 buckets. The dashed rectangle q is a query. (b) The bucket tree with tuple counts.

Fig. 1: An STHoles histogram.

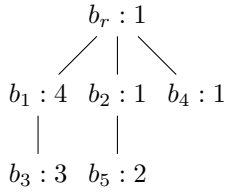
When estimating the number of tuples in q (Figure 1), STHoles computes the intersections with histogram buckets. q intersects with b_r and b_2. Using (1), we estimate the number of tuples in $b_r \cap q$ to be ≈ 0. The estimated number of tuples in $b_2 \cap q$ is a little less than 1.5. So the overall approximated number of tuples will be ≈ 1.5. We can see that the real number of tuples is 3.

Adding buckets. After the query is executed, the real numbers of tuples falling into $b_r \cap q$ and $b_2 \cap q$ become known. The histogram *refines* itself by *drilling* new buckets. The process consists of adding new buckets and updating the frequencies of existing buckets. Figure 2 shows the histogram with two buckets added. Because the intersection of $b_r \cap q$ is not a rectangle, it is shrunk across one dimension until it has a rectangular shape. Note that the frequencies of b_r and b_2 are also updated to reflect the new information about the tuple placement in the histogram.

Removing buckets. As Figure 2 has illustrated, STHoles adds buckets after a query is executed and its results are known. In our example, two new buckets,



(a) b_4 and b_5 are the newly added buckets.



(b) The bucket tree with b_4 and b_5. Note that the tuple counts of b_2 and b_r are adjusted as well.

Fig. 2: Inserting buckets into STHoles.

b_4 and b_5, have been added to the histogram. Buckets take up storage space, and storage space is limited. In order to accommodate new buckets, the histogram needs to compact itself.

STHoles frees up space by *merging* similar buckets. The idea is to assume that the histogram in its current form contains perfect information about the dataset, and to try to merge buckets so that the histogram changes as little as possible.

For a given merge, if the histogram before the merge is H, and it is H' after the merge, then the penalty μ of performing the merge (and thus substituting H with H') is

$$\mu(H, H') = \int_{u \in D} |est(u, H) - est(u, H')| du \quad (2)$$

where D is the data domain.

We write

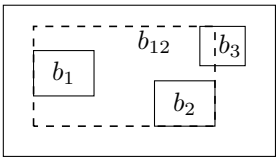
$$(b_1, b_2) \rightarrow b' \quad (3)$$

to denote that the bucket b' is the result of merging the buckets b_1 and b_2.

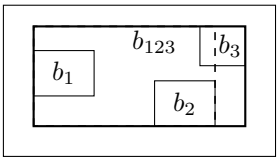
Among available merges, the one with the lowest penalty is picked. Each bucket can merge only with its parent bucket or its siblings. During the parent-child merge, the tuple count of the parent node is increased by the child node tuple count. Then, the child node is removed. The sibling-sibling merge is slightly more complicated. Figure 3 shows merging buckets b_1 and b_2. The minimal rectangle which contains both is computed. As it intersects with b_3 which also is a sibling of b_1 and b_2, the rectangle is extended to include b_3 as well. In this way, the bucket b_{123} on Figure 3 (b) is obtained. Its frequency is computed based on the frequency of the parent node, $n(b_1)$, $n(b_2)$ and $n(b_3)$. Then the new bucket is added, and b_1, b_2 are removed. Note that b_3 stays as a child bucket of b_{123}.

3 Problems with Self-Tuning

Let us now describe the problems associated with this self-tuning process. We focus on **Sensitivity to Learning** and **Stagnation**: we analyze how they occur and how initialization helps to avoid them. We only briefly stop on **Dimensionality**, as this problem is relatively well known to the histogram construction and clustering communities.



(a) The b_{12}, which encloses both b_1 and b_2, partially intersects with b_3.



(b) Thus, we extend it to fully include b_3 as well, obtaining the rectangle b_{123}.

Fig. 3: Merging two sibling buckets.

3.1 Sensitivity to Learning

Informally, Sensitivity to Learning is the phenomenon that changing the order of the learning queries has a significant impact on the histogram precision.

We will call the workloads W_1 and W_2 permutations of each other if they consist of same queries, but in different order. We will write $W_2 = \pi(W_1)$, where π is some permutation. Given a histogram H and a workload W, we will write $H|W$ to indicate the histogram which results from H after it learns the query feedback from W.

At first sight, histograms resulting from two workloads where one is a permutation of the other one, $H|W$ and $H|\pi(W)$, should produce very close estimates. We first show on an example how permutation of queries can result in histograms which considerably differ in structure.

Example 1. Figure 4 demonstrates what happens when we change the order of the queries. The bucket limit for the histogram is two buckets. Each row shows a sequence:

- the left column of the figure shows the order in which the queries arrive (denoted by numbers),
- the middle column is the situation after both queries are executed and buckets are drilled,
- the right column is the final configuration after one bucket is removed to meet the 2-bucket budget.

Clearly, the top-right histogram is better than the bottom-right one. It captures the data distribution well, while the one in the bottom-right misses out some tuples and has one bucket with regions of different densities.

Looking at the bucket-drilling procedure of STHoles, we can see why this happens. The histogram attempts to integrate the new information into the existing bucket structure, even if it means shrinking the new query rectangle. The rectangle corresponding to Query 1 in the bottom row is not a good bucket candidate because it contains regions with different densities. Nevertheless, it is added to the histogram. The second query (bottom row again) intersects with the first rectangle, and the resulting bucket is a shrunk version of the query. In other words, the

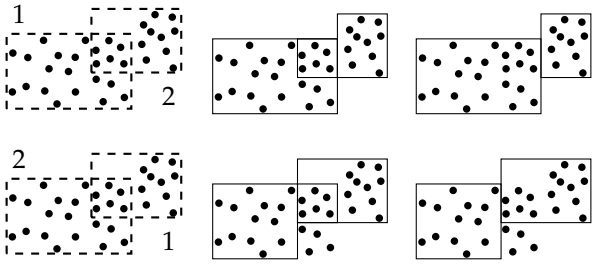


Fig. 4: Two queries and the histograms corresponding to them. Each row represents a different query order.

second query which brings along useful information about the tuple distribution is deformed (bottom row, middle), only because existing buckets are favored over new ones. □

In order to compare histograms we need a metric. A commonly used metric is the absolute error metric:

$$\varepsilon_D(H) = \int_{u \in D} |real(u) - est(u)| du \tag{4}$$

where $real(u)$ and $est(u)$ are the real and estimated cardinalities of point query u using the histogram H, and D is the attribute-value domain (D is usually fixed and we sometimes omit it for brevity). Low error means a better histogram.

Definition 1 (δ-sensitivity)**.** We call a histogram H δ-sensitive to learning w.r.t workload W if for some permutation π

$$|\varepsilon_D(H|W) - \varepsilon_D(H|\pi(W))| > \delta \tag{5}$$

δ-sensitivity means that changing the order of learning queries changes the estimation error by more than δ. Intuitively, if a workload W is informative (i.e., contains enough queries and does not miss out chunks of data), our expectation would be that a good histogram should not be very sensitive to workload permutations, that is, the δ should be small compared to $\varepsilon(H|W)$ and $\varepsilon(H|\pi(W))$.

3.2 Stagnation

Stagnation is the phenomenon of a histogram not being able to find a good bucket layout. There are several reasons why stagnation can occur. In this section we focus on one of the reasons, which is related to the "adaptiveness" of the histogram. We show that an adaptive histogram can get stuck in a locally optimal bucket layout and be unable to improve it even with an unlimited number of learning queries. This problem of getting stuck in local optima is commonplace among learning algorithms.

Stagnation occurs when the histogram does not have enough memory to detect certain clusters. It allocates the memory to other, unimportant data regions, due to the order of learning queries. As we will show, this can result in a situation when the histogram keeps the buckets in these unimportant regions and performs merges in important regions. Consequently, important clusters can remain undetected.

We show that in the process of *detecting* a bucket configuration which guarantees a certain estimation precision, we typically need more memory than what is needed to *store* a configuration with the same guarantee. The simplest example is a large, uniform, rectangular cluster, which requires only one bucket to store. However, we prove that it is impossible to detect this cluster using small rectangular queries if the memory budget is only one bucket.

We first give some auxiliary definitions, then define the notions of *Cluster Detectability Threshold* and of *Storage Threshold*. *Cluster Detectability Threshold* is the memory needed to detect the cluster. *Storage Threshold* is the memory needed to store the cluster so that the estimation error does not exceed a certain threshold.

Definition 2 (Histogram error on a cluster)**.** The estimation error of the histogram H on cluster C is defined as $\varepsilon_C(H)$ (equation (4)).

Definition 3 (Bucket Count)**.** The number of buckets in the histogram H is denoted by $b(H)$.

Definition 4 (Storage threshold)**.** A cluster C has storage threshold $\sigma(C, \beta)$ for error threshold β if it is possible to construct a histogram H using $\sigma(C, \beta)$ buckets such that

$$\varepsilon_C(H) \leq \beta$$

In other words, the storage threshold is the minimal number of buckets which allows to capture the distribution with error $\leq \beta$:

$$\sigma(C, \beta) = \min\{b(H) | \varepsilon_C(H) \leq \beta\} \tag{6}$$

Definition 5 (Cluster Detectability Threshold)**.** The detectability threshold of a cluster C for error threshold β, denoted by $\omega(C, \beta)$, is the minimal memory budget required to construct a histogram H such that

$$\varepsilon_C(H) \leq \beta$$

There is a caveat in the definition of Detectability Threshold. Namely, having $\omega(C, \beta)$ buckets only *makes possible* to detect C (in the sense that the error will be less than a given threshold). Having this much memory does not guarantee that the given algorithm will detect the cluster under a specific workload, it only says that there is at least one workload which makes possible to detect the cluster.

Lemma 1. *For any cluster C and error threshold β,*

$$\omega(C, \beta) \geq \sigma(C, \beta)$$

For all proofs, see [16].

In the forthcoming discussion we show that for several simple and common data distributions,

$\omega(C, \beta) > \sigma(C, \beta)$ for certain (relevant) values of β. Together with the result of Lemma 1, this allows us to state that detecting a cluster is never cheaper memory-wise than storing it, and very often it is more expensive.

We will assume that there is a root bucket in the histogram which encloses the entire dataspace, and that this bucket is fixed. When we say that the bucket limit is one bucket we mean it is one bucket plus this root.

Real-world data distributions differ significantly in their characteristics, as do the patterns by which the users query the data. The challenging scenario for self-tuning histograms is when the clusters are large, while the volume of the queries is relatively small compared to the volume of the clusters. We model this by assuming the queries are homogeneous – with unit volume, and the clusters which need to be detected are larger. We also assume that the queries and the clusters are aligned to a grid. Aligning queries and clusters makes cluster detection easier. Therefore, the problems related to cluster detection that we reveal here persist if we drop the assumption about grid-aligned queries and clusters.

Let the dataset be $[1, \ldots, N] \times [1, \ldots, N]$, then the query rectangles have the form $[i, j] \times [i + 1, j + 1]$ where $i, j \in \{1, \ldots, N - 1\}$.

Lemma 2. *C is a uniform cluster with dimensions $m \times k$. For this cluster, $\sigma(C, 0) = 1$ and $\omega(C, 0) = 2$. This means that*

1) *It possible to store the cluster using one bucket.*
2) *It is possible to detect the cluster using two buckets.*
3) *It is impossible to detect the cluster using only one bucket when $m \geq 2$ and $k \geq 2$. With one bucket, it is possible to detect a $m \times 1$ or a $1 \times k$ cluster.*

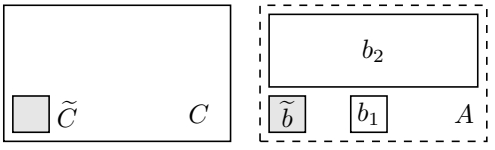
Definition 6 (Stagnation)**.** We say that a histogram stagnates at error level β with reducible error Δ if the histogram error is $\varepsilon(H) = \beta$, the error of H does not change by more than $\epsilon << \Delta$ by subsequent learning, and there exists a histogram H' with $b(H)$ buckets such that $\varepsilon(H') = \beta - \Delta$.

When the histogram stagnates but the reducible error is low, there is not much one can do. So when talking about stagnation, we mean the cases when Δ is comparable to β, say $\Delta \geq 0.3 \cdot \beta$.

Let us calculate the stagnation level of a uniform cluster of dimensions $m \times m$, $m \geq 2$. Lemma 2 states that at most a single row or a column of the cluster can be detected when $b(H) = 1$. Assuming that the cluster has unit density, the error of a histogram covering a row of the cluster is $(m-1) \cdot m$. In the meantime, the histogram H' with one bucket covering the cluster has error 0. Thus, H stagnates at error level $\beta = (m-1) \cdot m$ with reducible error $\Delta = \beta = (m - 1) \cdot m$.

Next, we discuss a scenario when stagnation occurs with a certain probability. The idea is that, once a part of the cluster has been detected, it does not merge with other parts, which prevents the detection of the whole cluster. This is related to sensitivity to learning: certain query orders result in a bucket configuration which cannot be improved by subsequent learning.

Again, we have a rectangular cluster C, of dimensions $k \times l$, $k, l \geq 3$. The cluster contains a dense "core" $\widetilde{C}$ of unit volume (Figure 5(a)). Except for the core, the cluster has unit density, the core density is γ.



(a) The cluster C with dense core $\widetilde{C}$.

(b) A partial bucket coverage for C.

Fig. 5: A cluster with a dense core, and a partial bucket coverage of it.

Lemma 3. *If the histogram contains a bucket covering exactly the core $\widetilde{C}$, and the core density $\gamma > 3$, then the cluster C is not detectable with budget $b(H) = 2$.*

The Lemmas 2 and 3 show two simple clusters on which the histogram stagnates if it does not have enough buckets for the cluster detection. The stagnation in Lemma 2 is unconditional. The stagnation in Lemma 3 is conditional on the workload: if the workload queries the cluster core, it cannot detect the whole cluster any more. The difference between the clusters in Lemma 2 and 3 is the dense sub-region in C. A bucket budget of 2 suffices for detection of a uniform cluster. When there is a dense sub-region in the cluster, which becomes detected, two buckets become insufficient to detect the cluster. This is because the dense sub-region does not merge with the rest of the cluster, and essentially the histogram has to detect C using only one bucket. This can be generalized to a case when the cluster has more than one dense sub-region – the more such dense sub-regions within a cluster, the harder it is to detect the cluster.

3.3 Dimensionality

Histogram construction in multi-dimensional spaces is a challenging problem [3]. In particular, finding locally correlated groups of tuples becomes harder as we increase the dimensionality of the space [23], [3]. One reason is that in higher-dimensional spaces not all attributes are relevant for a given region, i.e., some attributes are just random noise in certain regions, while they are relevant in others. This observation is named the curse of dimensionality [2].

Results from [2] directly imply that histogram construction becomes more difficult with increasing dimensionality, as follows: Let m be the dimensionality of the dataspace, P_m a sample of data points, Q_m a sample of query centers, and $dist$ a distance

measure. We define $DMIN_m = \min\{dist(P_m, Q_m)\}$, and $DMAX_m = \max\{dist(P_m, Q_m)\}$. $DMIN_m$ is the distance from a query center to the nearest data point, $DMAX_m$ is the distance to the farthest data point.

A lemma in [2] states that, if

$$\lim_{m\to\infty} var\frac{dist(P_m, Q_m)}{E[dist(P_m, Q_m)]} = 0 \quad (7)$$

then, for every $\varepsilon > 0$,

$$\lim_{m\to\infty} P[DMAX_m \leq DMIN_m \cdot (1+\varepsilon)] = 1 \quad (8)$$

Equation (8) says that, as the dimensionality of the dataspace increases, the distances to the nearest and to farthest points from a query center converge. In our terminology, given a cluster and a query center, a bucket with diameter $DMIN_m$ in all dimensions contains no points, and a bucket with diameter $DMAX_m$ in all dimensions contains all points of the cluster. As m grows, $DMIN_m$ and $DMAX_m$ become closer and closer. Full-dimensional buckets become "unstable", in the sense that slight variations in the bucket diameter can result in drastic change in the number of enclosed tuples. As we have seen in Section 3.1, self-tuning histograms are sensitive to the query order, that is, changing the order and the shape of the queries slightly can result in different bucket configurations. In particular, as Example 1 in Section 3.1 shows, changing the query order can affect the diameters of resulting buckets. Therefore, in high-dimensional space, full-dimensional buckets obtained via self-tuning are unlikely to accurately capture clusters. Subspace buckets, in turn, do not suffer from this problem, because the dimensionality of the buckets remains fixed – it does not depend on m. There are two considerations which are important in our context, both established in [2]. First, the condition in (7) holds for various real-life datasets and query workloads. Second, although (8) establishes convergence in the limit, the experiments in [2] show that for values of m as low as 15, $DMIN$ and $DMAX$ become sufficiently close.

Subspace clustering [17], [22], [20] tackles this problem by finding local attribute correlations in different projections of the data.

Finding lower-dimensional buckets is similar to finding subspace clusters. Subspace clustering methods access and analyze the entire data set. They specifically look for clusters in subspaces. In contrast, a self-tuning histogram looks only at the query feedback. This typically means that it only has limited information about the clusters it has to detect. Moreover, they do not deploy mechanisms which try to detect subspace clusters. As mentioned, subspace clusters appear as uniform when we increase the dimensionality of the dataspace. Therefore, we hypothesize that increased dimensionality of the dataspace will have negative impact on the quality of a self-tuning histogram, as subspace clusters will "dissolve" in full-dimensional space and become undetectable. We also hypothesize that initialization, coupled with self-tuning, enables the histogram to perform better on multi-dimensional data. We verify these hypotheses experimentally.

Initialization by subspace clusters does not enable us to use self-tuning histograms for datasets of arbitrary dimensionality. Multi-dimensional histograms (both static and self-tuning) do not scale well on datasets which have more than 4-5 dimensions. In the presence of this property, if one has a 100-dimensional dataset, there is no way around building several low-dimensional histograms and trying to combine them to obtain reasonable selectivity estimates [5]. Note however that histograms only need to be built on those attributes which appear in the query predicate (the WHERE clause). That is, on a 100-dimensional dataset where the query predicate refers to, say, four attributes, a 4-dimensional histogram is sufficient.

However, as our experiments and the example with the `Cars` relation show, even for datasets which are as low as 3-dimensional, the histogram can fail to accurately capture local correlations of attributes in projections of the data space. Subspace-clustering based initialization addresses this issue.

4 Subspace Clustering and Histogram Initialization

In this section, we describe our solution of histogram initialization with subspace buckets. Then, we show formally how this allows the histogram to become less sensitive to learning. In order to find a solution to the problems described above (Sensitivity to Learning, Stagnation, Dimensionality), we first discuss the shared *reasons* for those problems and a common *mechanism* in subspace initialization that solves these problems.

4.1 Initialization by Subspace Clusters

Self-tuning is able to *refine* the structure of the histogram. An uninitialized histogram has to rely on the first few queries to determine the top-level partitioning of the attribute-value space. If this partitioning is bad, the subsequent tuning is unlikely to "correct" it. The solution is to provide the histogram with a good initial configuration. This configuration should:

1) provide a top-level bucketing for the dataset, which can be later tuned using feedback;
2) capture the data regions in relevant dimensions, i.e., should exclude irrelevant attributes for each bucket.

We now describe how to initialize the histogram with subspace buckets. The subspace clustering algorithm finds dense clusters together with the set of relevant attributes. Then these clusters are transformed into histogram buckets.

Generally, clustering algorithms output clusters as a set of points. We need to transform this set of points into a rectangular representation. Cell-based clustering algorithms such as Mineclus [30] look for rectangular clusters.

Definition 7 (Bounding and Minimal Bounding rectangles)**.** A rectangle r is a bounding rectangle (BR) of a set of points P if

$$\forall p, (p \in P) \Rightarrow (\ p \in r)$$

$\tilde{r}$ is a Minimal Bounding Rectangle (MBR) for P, if for any bounding rectangle r of P, $\tilde{r} \subseteq r$.

We could take the MBR of a cluster as a bucket. However, we have found out in preliminary experiments that this has a drawback, which is illustrated in Figure 6. The cluster found on the left spans the whole y-dimension, therefore, it is one-dimensional. However, its MBR is two-dimensional (dashed rectangle on the right). In order to understand why the MBR on the right does not describe the cluster well, think of a set of 20 points uniformly distributed over the interval $[0, 1]$. If we take the a and b to be the minimum and the maximum of this set, respectively, we will obtain an interval $[a, b] \subset [0, 1]$, with cardinality 20 points. However, the distribution within $[a, b]$ would not be uniform, because the points were sampled from $[0, 1]$ and not $[a, b]$.

In two- and higher- dimensional spaces, taking the MBR can increase the dimensionality of the cluster, as demonstrated in Figure 6. This brings in an additional problem which is specific to histograms. The two-dimensional MBR introduces additional intersections with incoming query rectangles without measurable difference in estimation quality. This is undesirable.

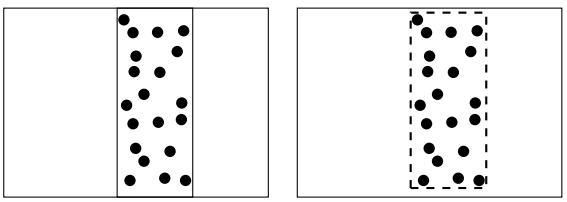


Fig. 6: On the left, the cluster found. On the right, the dashed rectangle is the MBR of the cluster. The solid rectangle on the left is the extended BR.

We can bypass this problem using the information produced by Mineclus. Mineclus outputs clusters as sets of tuples together with the relevant dimensions. Any cluster which does not use at least one of the dimensions of the dataspace is a subspace cluster. This means that the cluster spans $[min, max]$ on any unused dimension. To preserve subspace information, we introduce *extended BRs*.

Definition 8 (Extended BR)**.** Let cluster C consist of tuples $\{t_1, \ldots, t_n\}$ and dimensions $d_1, \ldots, d_k$. The extended BR of C is the minimal rectangle that contains the points $\{t_1, \ldots, t_n\}$ and spans $[min, max]$ for every dimension not in $d_1, \ldots, d_k$.

Definition 9 (Initialization by Subspace Clusters)**.** If the dataset consists of disjoint clusters $C_1, \ldots C_m$, then the initialized histogram is a histogram with buckets $\{b_1, \ldots, b_m\}$, where the bounding box of b_i is the extended BR of C_i, and the number of tuples in b_i is the tuple count of C_i.

A useful characteristic of Mineclus is that it assigns importance to clusters. The algorithm has a score function which decides whether a set of points is a cluster or not. The clusters themselves are then sorted according to this score. We found out that, if we use the important clusters as first queries in the initialization, we have a better estimation quality.

4.2 Initialization Analysis

Here we analyze how initialization by subspace clusters helps the histogram obtain a better structure with lower error.

4.2.1 Initialization and Sensitivity to Learning

We first conduct our analysis on a simplified scenario, then explain how more general cases can be reduced to this simple case.

Suppose that there is only one rectangular cluster, C, which has uniform density. The tuple density outside the cluster is considerably less than the cluster density. What matters in all the computations is the *density difference* between two regions, so without loss of generality we can set the outside density equal to 0.

First, we initialize the histogram with a bucket b_0 which has a bounding box that equals the bounding box of C. We call this histogram H_0. The error of histogram H_0 is $\varepsilon(H_0) = 0$ (ε is defined in Equation (4)). H_0 is depicted in Figure 7.

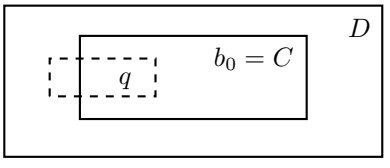


Fig. 7: The histogram H_0, with cluster C as a bucket. The dashed rectangle is the incoming query q.

Lemma 4. *For any workload W and $b(H) \geq 1$, the histogram $H_0|W$ will have zero error, that is, $\varepsilon(H_0|W) = 0$.*

This lemma shows that, once the bucket b_0 has been drilled, any sequence of queries cannot "spoil" the histogram structure. The bucket b_0 itself is *stable*, i.e., it does not disappear because there will always be better merge candidates (with less merge penalty). In terms of Definition 1, for any δ and any workload W, the initialized histogram is **not** δ-sensitive to learning.

This is because the estimation error is always 0, regardless of the workload.

On the other hand, if we start with no buckets at all, the histogram bucket structure will depend on the query order (see Example 1). If the workload consists of queries distributed uniformly, then the cluster may *eventually* get detected (if $b(H) \geq 2$). However, detecting the cluster can take a large number of queries to achieve, and in the meantime the histogram error will be > 0, and, again, will depend on the order of the queries.

We showed on a simple dataset with one dense cluster that capturing the cluster in a bucket makes the histogram insensitive to learning, for an arbitrary query workload and arbitrary number of maximum buckets allowed. In the absence of such a bucket, the histogram is sensitive to learning.

We can generalize Lemma 4 and show the same for a number of disjoint clusters $C_1, \ldots, C_l$. Next, the in-cluster density does not have to be constant, all that is needed is that the density drop from the cluster to outside the cluster is large enough to "discourage" the histogram from performing merges which include a bucket from within the cluster and one from outside.

In practice, of course, the data can be very complex. Having demonstrated here why initialization makes the histogram less sensitive to workloads for simple data distributions, we leave the complex cases for the experimental evaluation, and now turn to the issue of Stagnation.

4.2.2 Initialization and Stagnation

We have shown in Section 3.2 that detecting a cluster is never "cheaper" than storing it (Lemma 1), and that even for simple data distributions, the detectability threshold can be higher than the storage threshold (Lemmas 2 and 3). To store the clusters from Lemmas 2 and 3 optimally, the histogram has to allocate a bucket that corresponds to the cluster boundaries. Initialization helps to avoid stagnation by starting the histogram with hard-to-find buckets, which are the cluster boundaries.

A general analysis for arbitrary clusters and data distributions is hardly feasible. However, very often one is dealing with clusters which have smaller sub-clusters with significantly different density. We informally discuss how initialization improves detectability for data distributions which are noticeably more complex than the ones we discussed in Section 3.2. Suppose we have a rectangular cluster which has several dense sub-regions which we want to capture. Figure 8 shows a cluster with several dense sub-regions.

We argue why initialization is helpful in this case. We will discuss two separate scenarios.

1) The density of C is relatively low, and we do not need it as a bucket in the histogram.

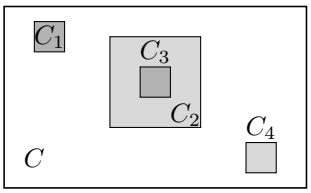


Fig. 8: Cluster C with several dense sub-regions.

2) The density of C is relatively high, and we need it as a bucket in order to have low error.

1. The density of C is low. In this case it is more important to use a bucket for one of the sub-clusters. The Mineclus algorithm finds dense clusters. The minimum density threshold is controlled by a parameter, and the clustering algorithm will discard clusters with low density.

2. The density of C is high. In this case we want to have C as a bucket in the histogram. We may or may not want to have all $C_1, \ldots C_4$ as separate buckets, depending on the density distribution elsewhere and the available memory. Lemmas 2 and 3 demonstrate that detecting the larger bounding box of the cluster with a limited amount of memory can be hard.

Initialization finds the boundaries of C. Further learning deals with the smaller clusters. If the clustering algorithm finds C, then it is likely to be a useful bucket in the histogram. Initialization allows to capture the cluster boundaries using only one bucket, which is memory-efficient. Self-tuning in contrast needs a larger memory budget to be able to detect C. If this budget is not available, the histogram stagnates.

5 Experiments

So far, we have described the problems with self-tuning approaches and our solution based on initialization using subspace clustering. In the following, we will empirically show that initial subspace clusters make self-tuning more accurate and robust. First, we show that our solution based on initialization provides a clear accuracy improvement over the uninitialized histogram (Section 5.2). Then we focus on the challenges from Section 3, namely – Sensitivity to Learning, Dimensionality, and Stagnation.

We do not compare our initialized histogram with static histograms. The reason is that our reference point is the uninitialized and sufficiently well-trained adaptive histogram. An extensive comparison of various histograms can be found in [29]. This includes comparison of static histograms with STHoles.

5.1 Experimental Setup

We have used two synthetic and one real-world data set. Dataset parameters are summarized in Table 1.

Synthetic data sets. The two synthetic datasets used are the *Cross* dataset and the *Gauss* dataset. The *Cross* dataset contains two one-dimensional clusters,

Dataset	Type	Dimensionality	Tuples
Cross	Synthetic	2	22,000
Gauss	Synthetic	6	110,000
Sky	Real-World	7	$\approx$ 1.7 million

TABLE 1: Dimensionalities and tuple counts of our datasets.

each cluster spanning a different dimension. Each cluster contains 10,000 tuples. Another 2,000 tuples are random noise. It is shown in Figure 9. The *Cross* dataset is very simple, in the sense that it is possible to perfectly describe it using 5 buckets. There can still be an estimation error due to randomly generated data (it is not perfectly uniform), but the error should be very low. It can be used to find out whether initialization makes any difference when the structure of the data is simple and a low estimation error is expected. The *Gauss* dataset is a 6-dimensional dataset. It contains subspace clusters which are multi-dimensional Gaussian bells, drawn in a k-dimensional subspace, $2 \leq k \leq 5$. The dimensions the clusters appear in are chosen at random. The dataset assigns 100,000 tuples to clusters, and 10,000 tuples are generated randomly as noise. The lower-dimensional clusters are easier to detect for the clustering and hard to detect using full-space queries. Figure 10 shows a 2-dimensional variant of the *Gauss* dataset. Here, the clusters themselves appear in the full-dimensional space.

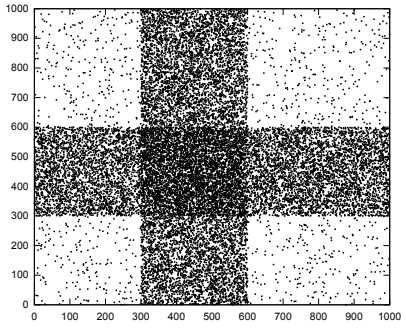


Fig. 9: The *Cross* dataset.

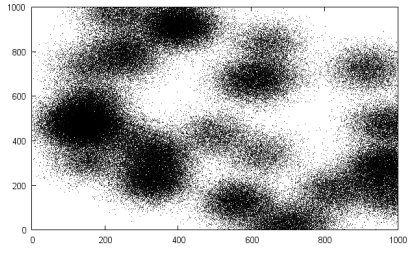


Fig. 10: A 2-dimensional variant of the *Gauss* dataset.

The Sloan Digital Sky Survey dataset. As a real-world dataset, we use one of the datasets published by the Sloan Digital Sky Survey (SDSS) [25]. It contains approximately 1,7 million tuples of observations. We call it the *Sky* dataset. We have removed the "class" column from the data, and the remaining dataset is 7-dimensional. The first two dimensions are the coordinates of the object in the sky, the next five columns contain brightness data – passed through different filters. Complex data correlations exist in the *Sky* dataset. There are several full-dimensional clusters, as well as subspace clusters in different projections of the data.

Queries, Buckets, Metrics. We generate queries which span a certain volume in the data space. The query centers either follow a uniform distribution, or they follow the data distribution, that is – are sampled from the data. If not stated explicitly, the query-center distribution is the uniform distribution. We also have conducted experiments with different workload-generation patterns, and the trends have been the same. Hence, we mostly stick to the pattern "random centers, fixed-volume queries" because this allows to compare results across experiments.

The precision of a histogram usually depends on the available space. We vary the number of histogram buckets from 50 to 250 like most other researchers do [27], [24], [3], [29].

To evaluate a histogram, we compare the cardinality estimates produced by the histogram to the real cardinality of the query. A cardinality estimate which is close to the real cardinality enables the optimizer to accurately estimate the costs of different plans, and to choose a good plan. Therefore, the quality of a histogram is conventionally measured by the error the histogram produces over a series of queries [3], [24], [27], [29], [12]. Given a workload W and histogram H, the *Mean Absolute Error* is:

$$E(H, W) = \frac{1}{|W|} \sum_{q \in W} |est(H, q) - real(q)| \qquad (9)$$

where $real(q)$ is the real cardinality of the query. In order for the results to be comparable across datasets, we normalize this error by dividing it by the error of a trivial histogram H_0 (cf. [3]). H_0 contains only one bucket which simply stores the number of tuples in the database.

$$NAE(H, W) = \frac{E(H, W)}{E(H_0, W)} \qquad (10)$$

Unless stated otherwise, the workload is the same for all histograms and contains 1,000 training and 1,000 simulation queries. The first 1,000 queries are only for training, and the error computation according to Equation (10) starts with the simulation queries.

5.2 Accuracy

In the first set of experiments we show that initialization improves estimation quality. Figures 11, 12 and 13 show the error comparison for the *Cross*, *Gauss* and *Sky* datasets. For all datasets, the initialized histogram outperforms the uninitialized version.

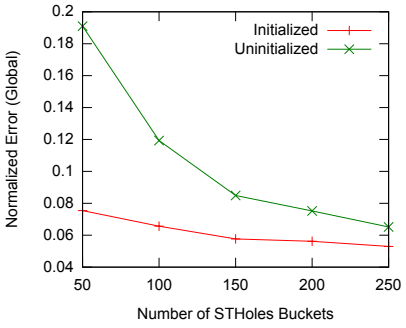


Fig. 11: Errors of initialized vs uninitialized histograms, $Cross[1\%]$ setting.

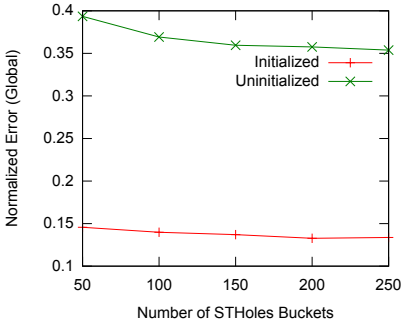


Fig. 12: Errors of initialized vs uninitialized histograms, $Gauss[1\%]$ setting.

As mentioned, the *Cross* dataset is simple and can easily be described with 5 to 6 buckets. Nevertheless, Figure 11 shows that initialization has a significant effect, improving the estimation accuracy. This is an experimental confirmation of the analysis conducted in Section 4.2. Initialization finds the 5-6 buckets which are essential for the good histogram structure, while a random workload of even 1,000 training queries is not enough for the uninitialized histogram to find this simple bucket layout. Figure 12 shows the error on the *Gauss* dataset, which contains more complex structures in the database in the form of Gaussian bells hidden in different projections of the data space. Comparing with the *Cross* dataset, we can see that the estimation error for both uninitialized and initialized histograms is higher. This is expected and is due to the fact that *Cross* is a piecewise-uniform dataset and *Gauss* is not. On the *Gauss* dataset we can see the effect of the subspace clustering much better, as the initialization now provides a considerably bigger benefit compared to the *Cross* dataset. Figure 13 shows the comparison on the *Sky* dataset. Here, the errors are higher than both for *Cross* and *Gauss* datasets. The benefit of subspace clustering is again clear: The initialized version has about half the error rate compared to the uninitialized version.

In all cases, the initialized histogram outperforms the uninitialized version. Moreover, for the *Gauss* and *Sky* datasets, the initialized histogram with only 50 buckets is significantly better than the uninitialized histogram with 250 buckets. Only on the simple *Cross* dataset the uninitialized histogram with 250 buckets

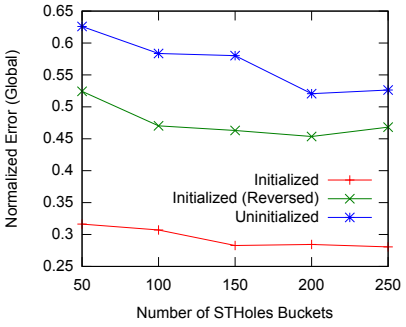


Fig. 13: Error comparison for $Sky[1\%]$ setting. The meaning of the green line "Initialized (Reversed)" is explained in Section 5.3.

reaches the quality of the initialized histogram with 50 buckets.

Clustering. The Mineclus algorithm uses parameters $\alpha, \beta, width$ to find clusters. α is used to determine the minimal density threshold of the cluster. If the fraction of tuples that fall into a region is less than α, the region is not considered a cluster. $width$ is used to determine the minimal width of the clusters, and β controls the importance of the cluster size vs its dimensionality. Our goal when choosing clustering parameters was to find dense clusters which are not too small. In our experiments, β did not have much influence on the clustering quality. We have varied α from 0.01 to 0.1. Higher values typically corresponded faster running times and worse initialization results.

alpha	*beta*	*width*	error	Clustering Time	Sim. time
0.01	0.10	10	0.27	502.40	199.94
0.05	0.10	10	0.37	87.16	191.30
0.10	0.10	10	0.45	29.40	198.06
0.01	0.30	10	0.31	411.00	205.60

TABLE 2: Some parameter values, errors and running times (in seconds) for the *Sky* dataset.

Table 2 shows running times and errors for some parameter values. The dataset is the *Sky* dataset. The histogram has 100 buckets. For a reference, the error of uninitialized STHoles on this dataset is 0.62. As the table shows, clustering times can differ significantly for different parameter values. However, we point out that we use clustering as a means of demonstrating that initialization is important for self-tuning methods. Mineclus can be optimized in several ways to utilize the fact that its output is used only for histogram initialization. First, we need only the cluster boundaries and approximate number of points in each cluster, not the exact clusters. Second, Mineclus attempts to assign all tuples to clusters, and usually a small fraction of points take the most time to decide which cluster to assign to. For our purposes, however, all tuples do not need to be assigned to clusters – all we need is approximate cluster boundaries. With such potential optimizations in mind, we conclude that the larger running times in Table 2 can be significantly

reduced if a specialized algorithm **based** on subspace clustering is used instead of Mineclus.

5.3 Robustness

We now revisit the challenges described in Section 3. The following experiments highlight the reasons why our initialized version outperforms traditional uninitialized histograms. Essentially, we investigate the sensitivity to learning and the effects of dimensionality of the data space on self-tuning.

Sensitivity to learning. To show that STHoles is sensitive to learning, we conducted experiments using permuted workloads as defined in Section 3. To show the effect of changing the order of queries, recall how we initialize the histogram. We generate rectangles with frequencies from the clustering output and feed this to the histogram in the order of importance. This importance is an additional output of the clustering algorithm. In the experiment in Figure 13, we use the same set of clusters to initialize the histogram, but in a reverse order of importance. Clearly, there is a significant difference between the normal initialization and the reverse one. This shows two things. First, permuting a workload changes the histogram error significantly (Sensitivity to Learning). Second, it shows the importance of the order of initialization, as the "correct order" has a noticeably lower error compared to the reversed order. Finally, Figure 14 shows

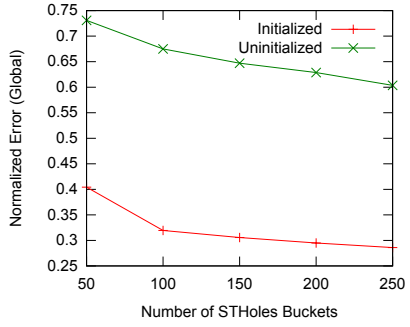


Fig. 14: Error comparison for $Sky[2\%]$ setting.

the *Sky* dataset with 2% volume queries instead of 1%. By comparing the results to the ones in Figure 13, we can see the effect of changed query volumes. Except for the case with 50 buckets, the error of the initialized version is essentially the same in both figures. Thus, the initialized version is considerably less sensitive to the change of query volume than the uninitialized histogram.

Dimensionality. To find out the effect of varying the dimensionality of the dataset, we have run experiments on 3, 4, and 5-dimensional variants of the *Cross* dataset. The idea has been to keep the density and the structure of the clusters constant, while varying the dimensionality of the cluster. An n-dimensional *Cross* dataset contains n clusters, each cluster is $n-1$ dimensional (analogous to the 2-dimensional *Cross* dataset). The dataset parameters are summarized in Table 3.

Dataset	Dimensionality	Tuples
Cross3d	3	9,000
Cross4d	4	360,000
Cross5d	5	13,500,000

TABLE 3: The higher-dimensional variants of the *Cross* dataset.

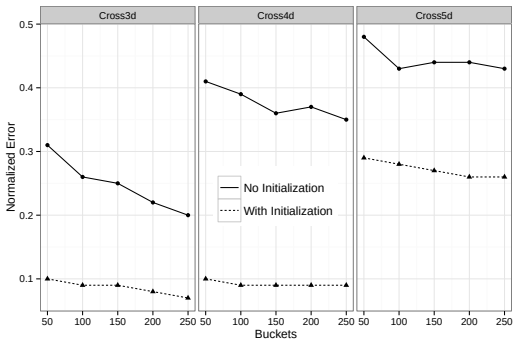


Fig. 15: Error comparison for Cross3d, Cross4d, and Cross5d datasets.

Figure 15 shows the errors of histograms, with and without initialization, on *Cross3d*, *Cross4d* and *Cross5d* datasets. A first thing to notice is that the error of the uninitialized histogram increases consistently by the same amount. The error of the initialized histogram stays the same for the three and four dimensional cases and is considerably lower that the uninitialized version.

For the five-dimensional *Cross5d* dataset the number of tuples has been too high, and the clustering algorithm could not handle the dataset with the parameter settings we had used for the lower-dimensional datasets (there was memory overflow). So we changed the clustering settings until we found settings which could handle the dataset. It is likely that this is the reason why the error of the initialized histogram here is higher than that of the 3 and 4 dimensional ones. It still is considerably lower than the error of the uninitialized histogram.

Running Mineclus on the *Sky* dataset, we have found 20 clusters, referred to as $\{C_0, \ldots, C_{19}\}$ subsequently. Out of those, 11 were full-dimensional and 9 were subspace clusters. Table 4 sums up the the information of the clusters in the 7-dimensional *Sky* dataset. We list the irrelevant dimensions that the clusters **do not** use for bucket representations (column "Unused"). Clearly, there are global structures detected in the full space. But we also have detected very specific correlations between some of the dimensions w.r.t. a subset of tuples in the database. We have conducted experiments with varying bucket counts, from 50 to 250, as follows. After every 100 queries (out of 2,000 total), we dump the histogram structure and look for subspace buckets. For all bucket counts, the uninitialized histogram has not created a

Cluster	Unused	Tuples	Cluster	Unused	Tuples
C_0	none	207,377	C_{10}	1	98,438
C_1	none	178,394	C_{11}	none	21,495
C_2	none	153,161	C_{12}	none	17,522
C_3	none	121,384	C_{13}	1, 2	153,311
C_4	none	114,699	C_{14}	1	17,437
C_5	none	83,026	C_{15}	1, 2	77,112
C_6	1	218,770	C_{16}	1, 2	39,799
C_7	none	54,760	C_{17}	1, 2, 7	21,913
C_8	none	50,846	C_{18}	1, 2, 3, 7	24,084
C_9	none	40,067	C_{19}	1, 2, 3, 5, 6	19,236

TABLE 4: Clusters found in the *Sky* dataset, the dimensions they do not use and number of tuples in each cluster.

single subspace bucket. The initialized version starts with several subspace buckets, which eventually are merged as the simulation goes on. The only case when subspace buckets are preserved through 2,000 queries is the initialized histogram with 250 buckets. We find this quite interesting, as the number of merges during 2,000 query-simulation with 250 buckets is very high, and 4 subspace clusters "survive" that many merges. With the initialized histogram we observe that the higher the number of buckets, the longer the subspace buckets survive.

Our conclusion from these dimensionality experiments is the following: STHoles is unable to find subspace buckets on its own. To do so, it needs initialization. Thus, our conjecture from Section 3.3 is true – subspace clusters are hard to find using the information provided by query feedback.

Stagnation. So far, we have used the same training workload both for initialized and uninitialized histograms. Looking at the *Sky* dataset in Figures 13 and 14, we can see that the uninitialized histogram has twice the error rate of the initialized version. One wonders whether additional training can help to overcome this difference. By providing extra training queries for the uninitialized version, we can find out whether the effects of initialization are temporary or persistent. The setup for this experiment is as follows:

1) Start by training both initialized and uninitialized histograms with the same 1,000 queries.
2) Continue the training of the uninitialized version with an additional 18,000 queries.
3) Evaluate both histograms using the same 1,000 query workload.

Figure 16 plots the errors of the uninitialized and heavily-trained as well as of the initialized histograms. The initialized version consistently outperforms the heavily-trained histogram. Comparing Figures 13 and 16, we see that the error rate of the heavily-trained version is actually a bit higher than that of the normally trained histogram (both are uninitialized histograms). The reason is twofold. First, due to *Stagnation*, extra training does not provide benefits after a certain number of queries. Second, there is variance due to different workloads – another manifestation of *Sensitivity to Learning*. This is

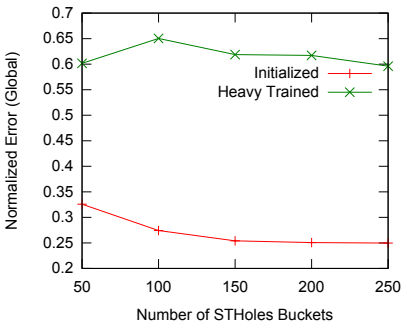


Fig. 16: Error comparison of heavily-trained vs Initialized histograms, *Sky*[1%] setting.

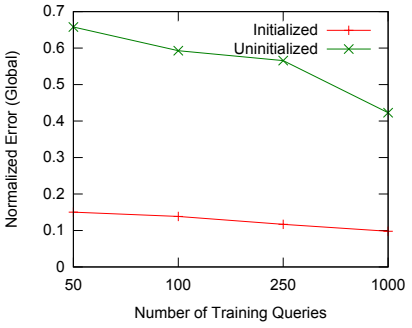


Fig. 17: Histogram errors for different amounts of training queries, *Cross4d*[1%], 100 buckets.

an experimental verification of the results obtained in Section 3.2. Histogram learning stagnates after a number of learning queries. We have observed this effect throughout datasets and different workloads.

Next, we vary the number of training from very little (50 queries) to normal (1,000 queries). The dataset is *Cross4d* , the bucket count is fixed to 100 and the amount of training queries is one out of {50, 100, 250, 1000}. We have altered the default STHoles behavior for this experiment. In all other experiments, histogram refinement continues during the simulation. In this experiment, the histogram stops learning after the training workload is executed. The error is calculated as before – the average of 1,000 simulation queries. Figure 17 shows the error of the initialized and uninitialized histograms. The reason we observe such a picture is that there are few, but well defined clusters in the *Cross4d* dataset. Initialization finds them, and further learning is essentially useless. In contrast, the uninitialized histogram benefits from training. However, 1,000 queries still are not enough to detect the four large clusters.

Additional Experiments. Additional experiments can be found in [16]. One of the experiments is on a 18-dimensional dataset from particle physics with 5 million tuples. Initialization reduces the error by 30%-50%. However, the simulations take considerably longer to complete than on lower-dimensional datasets.

Conclusion and remarks on Future Work can also be found in [16].

REFERENCES

[1] L. Baltrunas, A. Mazeika, and M. H. Böhlen. Multi-dimensional histograms with tight bounds for the error. In *IDEAS*, pages 105–112, 2006.

[2] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbors meaningful. In *Proc. ICDT*, pages 217–235, 1999.

[3] N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: a multidimensional workload-aware histogram. *SIGMOD Rec.*, 30:211–222, May 2001.

[4] S. Chaudhuri. An overview of query optimization in relational systems. In *In PODS*, 1998.

[5] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Found. Trends databases*, 4:1–294, Jan. 2012.

[6] A. Deshpande, M. Garofalakis, and R. Rastogi. Independence is good: dependency-based histogram synopses for high-dimensional data. *SIGMOD Rec.*, 30:199–210, May 2001.

[7] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. *SIGMOD Rec.*, 30:461–472, May 2001.

[8] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. *SIGMOD Rec.*, 29:463–474, May 2000.

[9] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM.

[10] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 275–286, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[11] I. T. Jolliffe. *Principal Component Analysis*. Springer, second edition, Oct. 2002.

[12] P. Karras and N. Mamoulis. Hierarchical synopses with optimal error guarantees. *ACM Trans. Database Syst.*, 33(3):18:1–18:53, Sept. 2008.

[13] A. Khachatryan and K. Boehm. Quantifying uncertainty in multi-dimensional cardinality estimations. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pages 1317–1320, New York, NY, USA, 2010. ACM.

[14] A. Khachatryan, E. Müller, K. Böhm, and J. Kopper. Efficient selectivity estimation by histogram construction based on subspace clustering. In *SSDBM*, pages 351–368, 2011.

[15] A. Khachatryan, E. Müller, C. Stier, and K. Böhm. Sensitivity of self-tuning histograms: Query order affecting accuracy and robustness. In *SSDBM*, pages 334–342, 2012.

[16] A. Khachatryan, E. Müller, C. Stier, and K. Böhm. Improving accuracy and robustness of self-tuning histograms by subspace clustering. Technical report, 2014. http://www.armsoft.am/research/KMSB-TR.pdf.

[17] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM TKDD*, 3(1):1–58, 2009.

[18] L. Lim, M. Wang, and J. S. Vitter. Sash: a self-adaptive histogram set for dynamically changing workloads. VLDB '2003, pages 369–380.

[19] J. Luo, X. Zhou, Y. Zhang, H. T. Shen, and J. Li. Selectivity estimation by batch-query based histogram and parametric method. In ADC '07, pages 93–102, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[20] E. Müller, S. Günnemann, I. Assent, and T. Seidl. Evaluating clustering in subspace projections of high dimensional data. *PVLDB*, 2(1):1270–1281, 2009.

[21] S. Muthukrishnan, V. Poosala, and T. Suel. On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. ICDT '99, pages 236–256, London, UK, 1999. Springer-Verlag.

[22] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: a review. *SIGKDD Explor. Newsl.*, 6(1):90–105, 2004.

[23] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. VLDB '97, pages 486–495, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[24] Y. J. Roh, J. H. Kim, Y. D. Chung, J. H. Son, and M. H. Kim. Hierarchically organized skew-tolerant histograms for geographic data objects. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 627–638, New York, NY, USA, 2010. ACM.

[25] SDSS Collaboration. Sloan Digital Sky Survey, August 27 2011.

[26] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A dynamic index for multi-dimensional objects. In *VLDB*, pages 507–518, 1987.

[27] U. Srivastava, P. J. Haas, V. Markl, M. Kutsch, and T. M. Tran. Isomer: Consistent histogram construction using query feedback. ICDE '06, pages 39–, Washington, DC, USA, 2006. IEEE Computer Society.

[28] H. Wang and K. C. Sevcik. A multi-dimensional histogram for selectivity estimation and fast approximate query answering. CASCON '03, pages 328–342. IBM Press, 2003.

[29] H. Wang and K. C. Sevcik. Histograms based on the minimum description length principle. *The VLDB Journal*, 17, May 2008.

[30] M. L. Yiu and N. Mamoulis. Frequent-pattern based iterative projected clustering. ICDM '03, pages 689–693.

Fig. 18: Andranik Khachatryan

Andranik Khachatryan has received his bachelor and master degrees from Yerevan State University and his PhD from Karlsruhe Institute of Technology (KIT), in 2012. Currently he is a lead specialist at Armsoft LLC, Research and Education group, and a lecturer in Armenian-Russian (Slavonic) University. His research interests include query optimization, selectivity estimation, and adaptive histograms. He is also involved in tuning large-scale database systems.

Fig. 19: Emmaneul Müller

Dr. Emmanuel Müller is a senior researcher and lecturer at the Karlsruhe Institute of Technology (KIT) and postdoctoral fellow at the University of Antwerp. He is leading a research group on data mining with focus on heterogeneous databases. His research interests cover efficient data mining in high dimensional, graph, and stream data, detection of clusters and outliers, and correlation analysis in heterogeneous databases. Further, he is leading the open-source initiative OpenSubspace ensuring repeatability and comparability of evaluations in recent data mining publications. Dr. Müller received his Diploma in 2007 and his PhD in 2010 from RWTH Aachen University. He has presented several tutorials at major data mining, database, and machine learning conferences, edited a special issue for the Machine Learning Journal, and organized workshops for ECML/PKDD'11, SDM'12, KDD'13, and KDD'14.

Fig. 20: Placeholder for Christian Stier

Christian Stier is a research scientist at the FZI Research Center for Information Technology and a PhD student at the Karlsruhe Institute of Technology. In 2014 he received the MSc degree in computer science at the Karlsruhe Institute of Technology, Germany. During his studies he focused on the fields of software engineering and data mining. His current research interests include software architecture evaluation with a focus on energy efficiency, self-adaptive software systems as well as software performance engineering.

Fig. 21: Klemens Böhm

Klemens Böhm is full professor (chair of databases and information systems) at Karlsruhe Institute of Technology (KIT), Germany, since 2004. Prior to that, he has been professor of applied informatics/data and knowledge engineering at University of Magdeburg, Germany, senior research assistant at ETH Zürich, Switzerland, and research assistant at GMD – Forschungszentrum Informationstechnik GmbH, Darmstadt, Germany. Current research topics at his chair are knowledge discovery and data mining in big data, data privacy and workflow management. Klemens gives much attention to collaborations with other scientific disciplines and with industry.