QuickSel: Quick Selectivity Learning with Mixture Models

Yongjoo Park, Shucheng Zhong, Barzan Mozafari

ABSTRACT

Estimating the selectivity of a query is a key step in almost any cost-based query optimizer. Most of today's databases rely on histograms or samples that are periodically refreshed by re-scanning the data as the underlying data changes. Since frequent scans are costly, these statistics are often stale and lead to poor selectivity estimates. As an alternative to scans, *query-driven histograms* have been proposed, which refine the histograms based on the actual selectivities of the observed queries. Unfortunately, these approaches are either too costly to use in practice—i.e., require an exponential number of buckets—or quickly lose their advantage as they observe more queries. For example, the state-of-the-art technique requires 318,936 buckets (and over 8 seconds of refinement overhead per query) after observing only 300 queries.

In this paper, we propose a *selectivity learning* framework, called QuickSel, which falls into the query-driven paradigm but does not use histograms. Instead, it builds an internal *model* of the underlying data, which can be refined significantly faster (e.g., only 1.9 milliseconds for 300 queries). This fast refinement allows QuickSel to continuously learn from *each query* and yield increasingly more accurate selectivity estimates over time. Unlike query-driven histograms, QuickSel relies on a mixture model and a new optimization algorithm for training its model. Our extensive experiments on two real-world datasets confirm that, given the same target accuracy, QuickSel is on average 254.6× faster than state-of-the-art query-driven histograms, including ISOMER and STHoles. Further, given the same space budget, QuickSel is on average 57.3% and 91.1% more accurate than periodically-updated histograms and samples, respectively.

ACM Reference Format:
Yongjoo Park, Shucheng Zhong, Barzan Mozafari. 2019. QuickSel: Quick Selectivity Learning with Mixture Models. In *Proceedings of ACM International Conference on Management of Data (SIGMOD 2019).* ACM, New York, NY, USA, Article 4, 18 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Estimating the *selectivity* of a query—the fraction of input tuples that satisfy the query's predicate—is a fundamental component in cost-based query optimization, including both traditional RDBMSs [3, 6, 9, 11, 77] and modern SQL-on-Hadoop engines [40, 82]. The estimated selectivities allow the query optimizer to choose the cheapest access path or query plan [52, 84].

Today's databases typically rely on histograms [3, 9, 11] or samples [77] for their selectivity estimation. These structures need to be populated in advance by performing costly table scans. However, as the underlying data changes, they quickly become stale and highly inaccurate. This is why they need to be updated periodically, creating additional costly operations in the database engine (e.g., `ANALYZE table`).[1]

To address the shortcoming of the scan-based approaches, numerous proposals for query-driven histograms have been proposed, which continuously correct and refine the histograms based on the actual selectivities observed after running each query [13, 14, 21, 51, 63, 70, 80, 86, 89]. There are two approaches to query-driven histograms. The first approach [13, 14, 21, 63], which we call *error-feedback histograms*, recursively splits existing buckets (both boundaries and frequencies) for every distinct query observed such that their error is minimized for the latest query. Since the error-feedback histograms do not minimize the (average) error across multiple queries, their estimates tend to be much less accurate.

To achieve a higher accuracy, the second approach is to compute the bucket frequencies based on the maximum entropy principle [51, 70, 80, 86]. However, this approach (which is also the state-of-the-art) requires solving an optimization problem, which quickly becomes prohibitive as the number of observed queries (and hence, number of buckets) grows. Unfortunately, one cannot simply prune the buckets in this approach, as it will break the underlying assumptions of their optimization algorithm (called *iterative scaling*, see Section 2.3 and Appendix B for details). Therefore, they prune the *observed queries* instead in order to keep the optimization overhead feasible in practice. However, this also

[1]Some database systems [11] automatically update their statistics when the number of modified tuples exceeds a threshold.

Table 1: The differences between query-driven histograms [51, 69, 70, 80, 86] and our method (QuickSel)

	Query-driven Histograms	**QuickSel (ours)**	**Our Contribution**
Model	histograms (non-overlapping buckets)	mixture models (overlapping subpopulations)	Employs a new expressive model → **no exponential growth of complexity**
Training	*maximum entropy* solved by *iterative scaling*	*min difference from a uniform distribution* solved *analytically*	A new optimization objective and its reduction to quadratic programming (solved analytically) → **fast training and model refinements**

means discarding data that could be used for learning a more accurate distribution.

Our Goal Our goal is to develop a new framework for selectivity estimation that can quickly refine its *model* after observing each query and, thereby, produce increasingly more accurate estimates over time. We call this new framework *selectivity learning*. In particular, we focus on designing a low-overhead method that can scale to a large number of observed queries without requiring an exponential number of buckets.

Our Model To overcome the limitations of query-driven histograms, we use a *mixture model* [19] to capture the unknown distribution of the data. A mixture model is a probabilistic model to approximate an arbitrary probability density function (pdf), say $f(x)$, using a combination of simpler pdfs:

$$f(x) = \sum_{z=1}^{m} h(z)\, g_z(x) \tag{1}$$

where $g_z(x)$ is the z-th simpler pdf and $h(z)$ is its corresponding weight. The subset of the data that follows each of the simpler pdfs is called a *subpopulation*. Since the subpopulations are allowed to overlap with one another, a mixture model is strictly more expressive than histograms. In fact, it is shown that mixture models can achieve a higher accuracy than histograms [25], which is also confirmed by our empirical study (Section 5.5). To the best of our knowledge, we are the first to propose a mixture model for selectivity estimation.[2]

Challenges Using a mixture model for selectivity learning requires finding optimal parameter values for $h(z)$ and $g_z(x)$; however, this optimization (a.k.a. training) is challenging for two reasons.

First, the training process aims to find parameters that maximize the model *quality*, defined as $\int Q(f(x))\,dx$ for some metric of quality Q (e.g., entropy). However, computing this integral is non-trivial for a mixture model since its subpopulations may overlap in arbitrary ways. That is, the combinations of m subpopulations can create 2^m distinct ranges, each with a potentially different value of $f(x)$. As a result, naïvely computing the quality of a mixture model quickly becomes intractable as the number of observed queries grows.

Second, the outer optimization algorithms are often iterative (e.g., iterative scaling, gradient descent), which means they have to repeatedly evaluate the model quality as they search for optimal parameter values. Thus, even when the model quality can be evaluated relatively efficiently, the overall training/optimization process can be quite costly.

Our Approach First, to ensure the efficiency of the model quality evaluation, we propose a new optimization objective. Specifically, we find the parameter values that minimize the $L2$ distance (or equivalently, *mean squared error*) between the mixture model and a uniform distribution, rather than maximizing the entropy of the model (as pursued by previous work [51, 69, 70, 80, 86]). As described above, directly computing the quality of a mixture model involves costly integrations over 2^m distinct ranges. However, when minimizing the $L2$ distance, the 2^m integrals can be reduced to only m^2 multiplications, hence greatly reducing the complexity of the model quality evaluation. Although minimizing the $L2$ distance is much more efficient than maximizing the entropy, these two objectives are closely related (see Appendix A for a discussion).

In addition, we adopt a non-conventional variant of mixture models, called a *uniform mixture model*. While uniform mixture models have been previously explored in limited settings (with only a few subpopulations) [27, 35], we find that they are quite appropriate in our context as they allow for efficient computations of the $L2$ distance. That is, with this choice, we can evaluate the quality of a model by only using min, max, and multiplication operations (Section 3.2). Finally, our optimization problem can be expressed as a standard *quadratic program*, which still requires an iterative procedure.

Therefore, to avoid the costly iterative optimization, we also devise an analytic solution that can be computed more efficiently. Specifically, in addition to the standard reduction

[2]Our mixture model is also different from *kernel density estimation* (KDE) [20, 34, 39], which is constructed by scanning the actual data, rather than analyzing observed queries.

(i.e., moving some of the original constraints to the objective clause as penalty terms), we completely relax the positivity constraints for $f(x)$, exploiting the fact that they will be naturally satisfied in the process of approximating the true distribution of the data. With these modifications, we can solve for the solution analytically by setting the gradient of the objective function to zero. This simple transformation speeds up the training by 1.5×–17.2×. In addition, since our analytic solution requires a constant number of operations, the training time is also consistent across different datasets and workloads.

Using these ideas, we have developed a first prototype of our selectivity learning proposal, called QuickSel, which allows for extremely fast model refinements. As summarized in Table 1, QuickSel differs from—and considerably improves upon—query-driven histograms [13, 51, 63, 70, 80, 86] in terms of both modeling and training (see Section 7 for a detailed comparison).

Contributions We make the following contributions:

1. We propose the first mixture model-based approach to selectivity estimation (Section 3).
2. We propose a new optimization objective, namely minimum difference from the uniform distribution, for efficient training of a mixture model (Section 4.1).
3. We show that the min-difference problem can be reduced to a quadratic program and present an efficient strategy for solving it (Section 4.2).
4. We conduct extensive experiments on two real-world datasets to compare QuickSel's performance and state-of-the-art selectivity estimation techniques (Section 5).

2 PRELIMINARIES

In this section, we first define relevant notations in Section 2.1 and then formally define the problem of *query-driven selectivity estimation* in Section 2.2. Next, in Section 2.3, we discuss the drawbacks of previous approaches.

2.1 Notations

Table 2 summarizes the notations used throughout this paper.

Set Notations T is a relation that consists of d real-valued columns $C_1, \ldots, C_d$.[3] The range of values in C_i is $[l_i, u_i]$ and the cardinality (i.e., row count) of T is $N{=}|T|$. The tuples in T are denoted by $x_1, \ldots, x_N$, where each x_i is a size-d vector that belongs to $B_0 = [l_1, u_1] \times \cdots \times [l_d, u_d]$. Geometrically, B_0 is the area bounded by a hyperrectangle whose bottom-left corner is $(l_1, \ldots, l_d)$ and top-right corner is $(u_1, \ldots, u_d)$. The size of B_0 can thus be computed as $|B_0|{=}(u_1{-}l_1){\times}\cdots{\times}(u_d{-}l_d)$.

[3]Handling integer and categorical columns is discussed in Section 2.2.

Table 2: Notations. pdf stands for probability density function; pmf stands for probability mass function.

Symbol	Meaning
T	a table (or a relation)
C_i	i-th column (or an attribute) of T; $i = 1, \ldots, d$
$\lvert T\rvert$	the number of tuples in T
$[l_i, u_i]$	the range of the values in C_i
x	a tuple of T
B_0	the domain of x; $[l_1, u_1] \times \cdots \times [l_d, u_d]$
P_i	i-th predicate
B_i	hyperrectangle range for the i-th predicate
$\lvert B_i\rvert$	the size (of the area) of B_i
$x \in B_i$	x belongs to B_i; thus, satisfies P_i
$I(\cdot)$	indicator function that returns 1 if its argument is true and 0 otherwise
s_i	the selectivity of P_i for T
(P_i, s_i)	i-th observed query
n	the total number of observed queries
$f(x)$	pdf of the tuple x (of T)

Predicates We use P_i to denote the (selection) predicate of the i-th query on T. In this paper, a predicate is a conjunction[4] of one or more *constraints*. Each constraint is a *range constraint*, which can be one-sided (e.g., $3 \leq C1$) or two-sided (e.g., $-3 \leq C1 \leq 10$). This range can be extended to also handle equality constraints on categorical data (see Section 2.2). Each predicate P_i is represented by a hyperrectangle B_i. For example, a constraint "$1 \leq C1 \leq 3$ AND $2 \leq C2$" is represented by a hyperrectangle $(1, 3) \times (2, u2)$, where $u2$ is the upper-bound of $C2$. We use P_o to denote an empty predicate, i.e., one that selects all tuples.

Selectivity The *selectivity* s_i of P_i is defined as the *fraction* of the rows of T that satisfy the predicate. That is, $s_i = (1/N) \sum_{k=1}^{N} I(x_k \in B_i)$, where $I(\cdot)$ is the indicator function. A pair (P_i, s_i) is referred to as an *observed query*.[5] Without loss of generality, we assume that n queries have been observed for T and seek to estimate s_{n+1}. Finally, we use $f(x)$ to denote the joint probability density function of tuple (that has generated tuples of T).

2.2 Problem Statement and Supported Queries

Next, we formally state the problem of query-driven selectivity estimation:

Problem 1 (Query-driven Selectivity Estimation) *Consider a set of n observed queries* $(P_1, s_1), \ldots, (P_n, s_n)$ *for* T. *By*

[4]See Section 2.2 for a discussion of disjunctions and negations.
[5]This pair is also referred to as an *assertion* by prior work [79].

definition, we have the following for each $i = 1, \ldots, n$:

$$\int_{x \in B_i} f(x)\, dx = s_i$$

Then, our goal is to build a model of $f(x)$ that can estimate the selectivity s_{n+1} of a new predicate P_{n+1}.

Initially, before any query is observed, we can conceptually consider a default query $(P_0, 1)$, where all tuples are selected and hence, the selectivity is 1 (i.e., no predicates).

Discrete and Categorical Values Problem 1 can be extended to support discrete attribute (e.g., integers, characters, categorical values) and equality constraints on them, as follows. Without loss of generality, suppose that C_i contains the integers in $\{1, 2, \ldots, b_i\}$. To apply the solution to Problem 1, it suffices to (conceptually) treat these integers as real values in $[1, b_i + 1]$ and then convert the original constraints on the integer values into range constraints as follow. A a constraint of the form "$C_i = k$" will be converted to a range constraint of the form $k \le C_i \le k + 1$. Mathematically, this is equivalent to replacing a probability mass function with a probability density function defined using *dirac delta functions*.[6] Then, the summation of the original probability mass function can be converted to the integration of the probability density function. String data types (e.g., char, varchar) and their equality constraints can be similarly supported, by conceptually mapping each string into an integer (preserving their order) and applying the conversion described above for the integer data type.

Supported Queries Similar to previous work [13, 21, 51, 63, 69, 70, 80, 86], our technique supports selectivity estimation for predicates with conjunctions, negations, and disjunctions of range and equality constraints on numeric and categorical columns. In other words, we currently do not support wildcard constraints (e.g., `LIKE '*word*'`), `EXISTS` constraints, and `ANY` constraints. In practice, often a *fixed selectivity* is used for unsupported predicates, e.g., 3.125% in Oracle [77].

To simplify our presentation, we focus on conjunctive predicates. However, negations and disjunctions can also be easily supported. This is because our algorithm only requires the ability to compute the intersection size of pairs of predicates P_i and P_j, which can be done by converting $P_i \wedge P_j$ into a disjunctive normal form and then using the inclusion-exclusion principle to compute its size.

As in the previous work, we focus our presentation on predicates on a single relation. However, any selectivity estimation technique for a single relation can be applied to estimating selectivity of a join query whenever the predicates on the individual relations are independent of the join conditions [9, 40, 84, 91].

[6]A dirac delta function $\delta(x)$ outputs ∞ if $x = 0$ and outputs 0 otherwise while satisfying $\int \delta(x)\, dx = 1$.

2.3 Limitations of Query-driven Histograms

Here, we briefly describe how query-driven histograms work [13, 21, 51, 63, 70, 80, 86] and then discuss their limitations, which motivate our work.

How Query-driven Histograms Work To approximate $f(x)$ (defined in Problem 1), query-driven histograms adjust their bucket boundaries and bucket frequencies according to the queries they observe. Specifically, they first determine the bucket boundaries (*bucket creation* step). and then compute the frequencies of those buckets (*training* step), as described next.

1. *Bucket Creation*: Query-driven histograms determine their bucket boundaries based on the given predicate's ranges [13, 51, 70, 80, 86]. If the range of a later predicate overlaps with that of an earlier predicate, they split the bucket(s) created for the earlier one into two or more buckets in order to ensure that the buckets do not overlap with one another. Figure 1 shows an example of this bucket splitting operation.
2. *Training*: After creating the buckets, query-driven histograms assign frequencies to those buckets. Early work [13, 63] determines bucket frequencies in the process of bucket creations. That is, when a bucket is split into two or more, the frequency of the original bucket is also split (or adjusted) such that it minimizes the estimation error for the latest observed query.
   However, since this process does not minimize the (average) error across multiple queries, their estimates are much less accurate. More recent work [51, 70, 80, 86] has addressed this limitation by by explicitly solving an optimization problem based on the maximum entropy principle. That is, they search for bucket frequencies that maximize the *entropy* of the distribution while remaining consistent with the actual selectivities observed.

Although using the maximum entropy principle will lead to highly accurate estimates, it still suffers from two key limitations.

Limitation 1: Exponential Number of Buckets Since existing buckets may split into multiple ones for each new observed query, the number of buckets can potentially grow exponentially as the number of observed queries grows. For example, in our experiment in Section 5.5, the number of buckets was 22,370 for 100 observed queries, and 318,936 for 300 observed queries. Unfortunately, the number of buckets directly affects the training time. Specifically, *iterative*

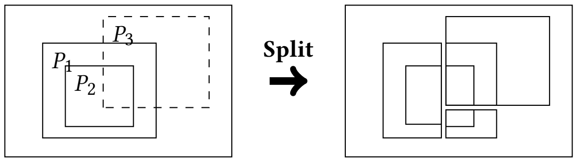


Figure 1: Bucket creation for query-driven histograms. P_3 is the range of a new predicate. The existing buckets (for P_1 and P_2) are split into multiple buckets. The total number of buckets may grow *exponentially* as more queries are observed.

scaling—the optimization algorithm used by all previous work [51, 69, 70, 80, 86]— the cost of each iteration grows linearly with the number of variables (i.e., the number of buckets). This means that the cost of each iteration can grow exponentially with the number of observed queries.

As stated in Section 1, we address this problem by employing a *mixture model*, which can express a probability distribution much more effectively than query-driven histograms. Specifically, our empirical study in Section 5.5 shows that—using the same number of parameters—a mixture model achieves considerably more accurate estimates than histograms.

Limitation 2: Non-trivial Bucket Merge/Pruning Given that query-driven histograms [70, 86] quickly become infeasible due to their large number of buckets, one might consider merging or pruning the buckets in an effort to reduce their training times. However, merging or pruning the histogram buckets violates the assumption used by their optimization algorithms, i.e., iterative scaling. Specifically, iterative scaling relies on the fact that a bucket is either *completely included* in a query's predicate range or *completely outside* of it.[7] That is, no partial overlap is allowed. This property must hold for each of the n predicates. However, merging some of the buckets will inevitably cause partial overlaps (between predicate and histogram buckets). For interested readers, we have included a more detailed explanation of why iterative scaling requires this assumption in Appendix B.

3 QUICKSEL: MODEL DEFINITION & SELECTITY ESTIMATION

This section presents how QuickSel models the population distribution and estimates the selectivity of a new query. QuickSel's model relies on a probabilistic model called a *mixture model*. In Section 3.1, we describe the mixture model employed by QuickSel. Section 3.2 describes how to estimate the selectivity of a query using the mixture model. Section 3.3 describes the details of QuickSel's mixture model construction.

[7]For example, this property is required for the transition from Equation (6) to Equation (7) in [70].

3.1 Uniform Mixture Model

A mixture model is a probabilistic model that expresses a (complex) probability density function (of the population) as a combination of (simpler) probability density functions (of *subpopulations*). The population distribution is the one that generates the tuple x of T. The subpopulations are internally managed by QuickSel to best approximate $f(x)$.

Uniform Mixture Model QuickSel uses a type of mixture model, called *the uniform mixture model.* The uniform mixture model represents a population distribution $f(x)$ as a weighted summation of multiple uniform distributions, $g_z(x)$ for $z = 1, \ldots, m$. Specifically,

$$f(x) = \sum_{z=1}^{m} h(z)\, g_z(x) = \sum_{z=1}^{m} w_z\, g_z(x) \tag{2}$$

where $h(z)$ is a categorical distribution that determines the weight of the z-th subpopulation, and $g_z(x)$ is the probability density function (which is a uniform distribution) for the z-th subpopulation. The support of $h(z)$ is the integers ranging from 1 to m; $h(z) = w_z$. The support for $g_z(x)$ is represented by a hyperrectangle G_z. Since $g_z(x)$ is a uniform distribution, $g_z(x) = 1/|G_z|$ if $x \in G_z$ and 0 otherwise. The locations of G_z and the values of w_z are determined in the training stage (Section 4). In the remainder of this section (Section 3), we assume that G_z and w_z are given.

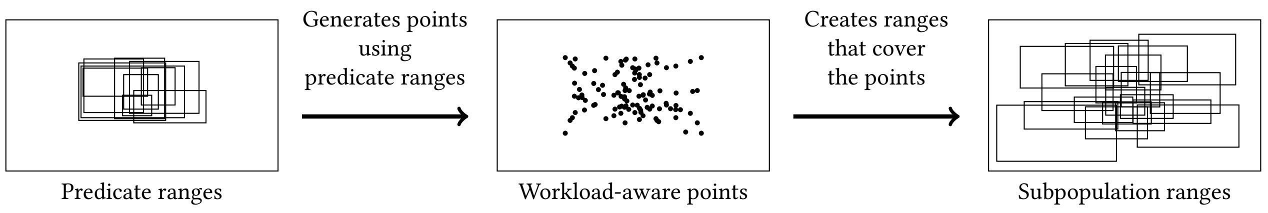
Benefit of Uniform Mixture Model The uniform mixture model was studied early in the statistics community [27, 35]; however, recently, a more complex model called *the Gaussian mixture model* has received more attention [19, 78, 101].[8] The Gaussian mixture model uses a Gaussian distribution for each subpopulation; the smoothness of its probability density function (thus, differentiable) makes the model more appealing when gradients need to be computed. Nevertheless, we *intentionally* use the uniform mixture model for QuickSel due to its computational benefit in the training process, as we describe below.

As will be presented in Section 4.2, QuickSel's training involves the computations of the intersection size between two subpopulations, for which the essential operation is evaluating the following integral:

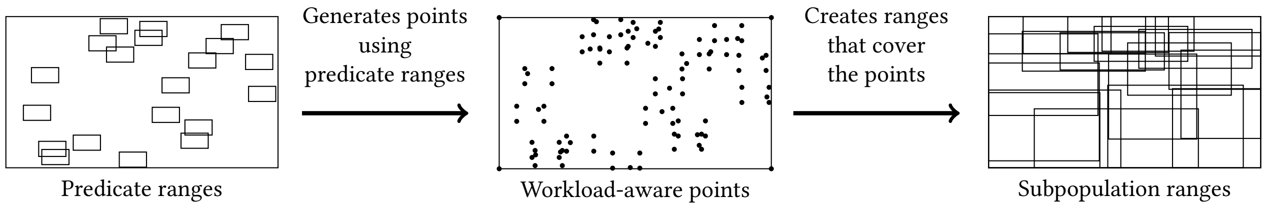
$$\int g_{z1}(x)\, g_{z2}(x)\; dx$$

Evaluating the above expression for multivariate Gaussian distributions, e.g., $g_{z1}(x) = \exp\left(-x^\top \Sigma^{-1} x\right) / \sqrt{(2\pi)^d\, |\Sigma|}$, requires numerical approximations [31, 49], which are either slow or inaccurate. In contrast, the intersection size between

[8]There are other variant mixture models [17, 73], which we do not discuss in this work.



(a) Case 1: Highly-overlapping query workloads



(b) Case 2: Scattered query workloads

Figure 2: QuickSel's subpopulation creation. Due to the property of mixture model (i.e., subpopulations may overlap with one another), creating subpopulations is straightforward for diverse query workloads.

two hyperrectangles can be exactly computed by simple min, max, and multiplication operations.

3.2 Selectivity Estimation with UMM

For the uniform mixture model, computing the selectivity of a predicate P_i is straightforward:

$$\int_{B_i} f(x)\,dx = \int_{B_i} \sum_{z=1}^{m} w_z\, g_z(x)\,dx = \sum_{z=1}^{m} w_z \int_{B_i} g_z(x)\,dx$$
$$= \sum_{z=1}^{m} w_z \int \frac{1}{|G_z|} I(x \in G_z \cap B_i)\,dx = \sum_{z=1}^{m} w_z \frac{|G_z \cap B_i|}{|G_z|}$$

Recall that both G_z and B_i are represented by hyperrectangles. Thus, their intersection is also a hyperrectangle, and computing its size is straightforward.

3.3 Subpopulations from Observed Queries

We describe QuickSel's approach to determining the boundaries of G_z for $z = 1, \ldots, m$. Note that how to determine G_z is orthogonal to the model training process, which we describe in Section 4; thus, even if one devises an alternative approach to creating G_z, our fast training method is still applicable.

QuickSel creates m hyperrectangular ranges[9] (for the supports of its subpopulations) in a way that satisfies the following simple criteria: if more predicates involve a point x, use a larger number of subpopulations for x. Unlike query-driven histograms, QuickSel can easily pursue this goal by exploiting the property of a mixture model: the supports of subpopulations may overlap with one another.

In short, QuickSel generates multiple points (using predicates) that represent the query workloads and create hyperrectangles that can sufficiently cover those points. Specifically, QuickSel performs the following operations for creating G_z for $z = 1, \ldots, m$.

1. Within each predicate range, generate multiple random points r. Generating a large number of random points increases the consistency (by lowering the randomness); however, QuickSel limits the number to 10 since generating more than 10 points did not improve accuracy in our preliminary study.
2. Use simple random sampling to reduce the number of points to m, which serves as the centers of G_z for $z = 1, \ldots, m$.
3. The size of each G_z is determined in a way that it slightly overlaps with its neighbor subpopulations (thus, covering all generated r). QuickSel obtains the size of G_z by averaging the distances to the 10 closest centers (of $G_{z'}$ where $z' \neq z$).

Figure 2 illustrates how the subpopulations are created using both (1) highly-overlapping query workloads and (2) scattered query workloads. In both cases, QuickSel generates random points to represent the distribution of query workloads, which is then used to create G_z ($z = 1, \ldots, m$), i.e., the supports of subpopulations. The following section describes how to assign the weights (i.e., $h(z) = w_z$) of these subpopulations.

[9]The number m of subpopulations is set to $\min(4 \cdot n, 4{,}000)$, by default.

4 QUICKSEL: MODEL TRAINING

This section describes how to compute the weights w_z of QuickSel's subpopulations. For training its model, QuickSel finds the model that maximizes uniformity while being consistent with the observed queries. In Section 4.1, we formulate an optimization problem based on this criteria. Next, in Section 4.2, we present how to solve the optimization problem efficiently.

4.1 Training as Optimization

This section formulates an optimization problem for QuickSel's training. Let $g_0(x)$ be the uniform distribution with support B_0; that is, $g_0(x) = 1/|B_0|$ if $x \in B_0$ and 0 otherwise. QuickSel aims to find the model $f(x)$ such that the difference between $f(x)$ and $g_0(x)$ is minimized while being consistent with the observed queries.

There are many metrics that can measure the distance between two probability density functions $f(x)$ and $g_0(x)$, such as the earth mover's distance [83], Kullback-Leibler divergence [59], the mean squared error (MSE), the Hellinger distance, and more. Among them, QuickSel uses MSE (which is equivalent to $L2$ distance between two distributions) since it enables the reduction of our originally formulated optimization problem (presented shortly; Problem 2) to a quadratic programming problem, which can be solved efficiently by many off-the-shelf optimization libraries [2, 7, 8, 16]. Also, minimizing MSE between $f(x)$ and $g_0(x)$ is closely related to maximizing the entropy of $f(x)$ [51, 70, 80, 86]. See Appendix A for the explanation of this relationship.

MSE between $f(x)$ and $g_0(x)$ is defined as follows:

$$\mathrm{MSE}(f(x), g_0(x)) = \int (f(x) - g_0(x))^2 \; dx$$

Recall that the support for $g_0(x)$ is B_0. Thus, QuickSel obtains the optimal weights by solving the following problem.

Problem 2 (QuickSel's Training) *QuickSel obtains the optimal parameter* $\boldsymbol{w}$ *for its model by solving:*

$$\operatorname*{arg\,min}_{\boldsymbol{w}} \int_{x \in B_0} \left(f(x) - \frac{1}{|B_0|} \right)^2 dx \tag{3}$$

$$\textit{such that} \int_{B_i} f(x) \; dx = s_i \quad \textit{for } i = 1, \ldots, n \tag{4}$$

$$f(x) \geq 0 \tag{5}$$

Equation (5) *ensures that* $f(x)$ *is a proper probability density function.*

4.2 Efficient Optimization

We first describe the challenges in solving Problem 2. Then, we describe how to overcome the challenges.

Challenge Solving Problem 2 in a naïve way is computationally intractable. For example, consider a mixture model consisting of (only) two subpopulations represented by G_1 and G_2, respectively. Then, $\int_{x \in B_0} (f(x) - g_0(x))^2 \, dx$ is:

$$\begin{aligned} &\int_{x \in G_1 \cap G_2} \left(\frac{w_1 + w_2}{|G_1 \cap G_2|} - g_0(x) \right)^2 dx \\ &+ \int_{x \in G_1 \cap \neg G_2} \left(\frac{w_1}{|G_1 \cap \neg G_2|} - g_0(x) \right)^2 dx \\ &+ \int_{x \in \neg G_1 \cap G_2} \left(\frac{w_2}{|\neg G_1 \cap G_2|} - g_0(x) \right)^2 dx \\ &+ \int_{x \in \neg G_1 \cap \neg G_2} \left(\frac{0}{|\neg G_1 \cap \neg G_2|} - g_0(x) \right)^2 dx \end{aligned}$$

Observe that with this approach, we need four separate integrations only for two subpopulations. In general, the number of integrations is $O(2^m)$, which is $O(2^n)$. Thus, this direct approach is computationally intractable.

Conversion One: Quadratic Programming We can solve Problem 2 efficiently by exploiting the property of the distance metric of our choice (i.e., MSE) and the fact that we use uniform distributions for subpopulations (i.e., UMM). The following theorem presents the efficient approach.

Theorem 1 *The optimization problem in Problem 2 can be solved by the following quadratic optimization:*

$$\begin{aligned} \operatorname*{arg\,min}_{\boldsymbol{w}} \quad & \boldsymbol{w}^\top Q \, \boldsymbol{w} \\ \textit{such that} \quad & A\boldsymbol{w} = \boldsymbol{s}, \quad \boldsymbol{w} \succcurlyeq \boldsymbol{0} \end{aligned}$$

where

$$(Q)_{ij} = \frac{|G_i \cap G_j|}{|G_i||G_j|} \qquad (A)_{ij} = \frac{|B_i \cap G_j|}{|G_j|}$$

The bendy inequality sign ($\succcurlyeq$) *means that every element of the vector on the left-hand side is equal to or larger than the corresponding element of the vector on the right-hand side.*

The proof of this theorem is in Appendix C. The implication of the above theorem is significant: we could reduce the problem of $O(2^n)$ complexity to the problem of only $O(n^2)$ complexity.

Conversion Two: Moving Constraints The quadratic programming problem in Theorem 1 can be solved efficiently by most off-the-shelf optimization libraries; however, we can solve the problem even faster by converting the problem to an alternative form. We first present the alternative problem, and then, we discuss it.

Problem 3 (QuickSel's QP) *QuickSel solves this problem alternative to the quadratic programming problem in Theorem 1:*

$$\arg\min_{\boldsymbol{w}} \quad \ell(\boldsymbol{w}) = \boldsymbol{w}^\top Q\, \boldsymbol{w} + \lambda\, \|A\boldsymbol{w} - \boldsymbol{s}\|^2$$

where λ is a large real value (QuickSel uses $\lambda = 10^6$).

In formulating Problem 3, two types of conversions are performed: (1) the consistency with the observed queries (i.e., $A\boldsymbol{w} = \boldsymbol{s}$) is moved into the optimization objective as a penalty term, and (2) the positivity of $f(x)$ is not explicitly specified (by $\boldsymbol{w} \succcurlyeq \boldsymbol{0}$). These two types of conversions have little impact on the solution for two reasons. First, to guarantee the consistency, a large penalty (i.e., $\lambda = 10^6$) is used. Second, the mixture model $f(x)$ is bound to approximate the true distribution, which is always non-negative. We empirically examine the advantage of solving Problem 3 (instead of solving the problem in Theorem 1 directly) in Section 5.4.

The solution $\boldsymbol{w}^*$ to Problem 3 can be obtained in a straightforward way by setting its gradients of the objective (with respect to $\boldsymbol{w}$) equal to $\boldsymbol{0}$:

$$\frac{\partial \ell(\boldsymbol{w}^*)}{\partial \boldsymbol{w}} = 2\, Q\, \boldsymbol{w}^* + 2\, \lambda\, A^\top (A\, \boldsymbol{w}^* - \boldsymbol{s}) = \boldsymbol{0}$$
$$\Rightarrow \boldsymbol{w}^* = \left(Q + \lambda\, A^\top A\right)^{-1} \lambda\, A\, \boldsymbol{s}$$

Observe that $\boldsymbol{w}^*$ is expressed in a closed form; thus, we can obtain $\boldsymbol{w}^*$ analytically instead of using iterative procedures typically required for general quadratic programming.

5 EXPERIMENT

In this section, we empirically study QuickSel. In summary, our results show the following:

1. **End-to-end comparison against other query-driven methods:** QuickSel was significantly faster (254.6× on average) for the same accuracy—and much more accurate estimates (46.8%–75.3% lower error) for the same time limit—than previous query-driven methods. (Section 5.2)
2. **Comparison against periodic database scans:** For the same storage size, QuickSel's selectivity estimates were 77.7% and 91.3% more accurate than scan-based histograms and sampling, respectively. (Section 5.3)
3. **Optimization efficiency:** QuickSel's optimization approach (Problem 3) was 1.5×–17.2× faster than solving the standard quadratic programming. (Section 5.4)
4. **Effectiveness of QuickSel's mixture model:** QuickSel's model produced considerably more accurate estimates than histograms given the same number of parameters. (Section 5.5)
5. **Sensitivity to different parameters:** QuickSel produced accurate selectivity estimates under various settings, such as different data correlation, workload-shift patterns, number of model parameters, and data dimensions. (Section 5.6)

5.1 Experimental Setup

Methods Our experiments compare QuickSel to six other selectivity estimation methods.

Query-driven Methods:

1. STHoles [21]: This method creates histogram buckets by partitioning existing buckets (as in Figure 1). The frequency of an existing bucket is distributed uniformly among the newly created buckets.
2. ISOMER [86]: This method applies STHoles for histogram bucket creations, but it computes the optimal frequencies of the buckets by finding the maximum entropy distribution. Among existing query-driven methods, ISOMER produced the highest accuracy in our experiments.
3. ISOMER+QP: This method combines ISOMER's approach for creating histogram buckets and QuickSel's quadratic programming (Problem 3) for computing the optimal bucket frequencies.
4. QueryModel [15]: This method computes the selectivity estimate by a weighted average of the selectivities of observed queries. The weights are determined based on the similarity of the new query and each of the queries observed in the past.

Scan-based Methods:

5. AutoHist: This method creates an equiwidth multidimensional histogram by scanning the data. It also updates its histogram whenever more than 20% of the data changes (this is the default setting with SQL Server's `AUTO_UPDATE_STATISTICS` option [10]).
6. AutoSample: This method relies on a uniform random sample of data to estimate selectivities. Similar to AutoHist, AutoSample updates its sample whenever more than 10% of the data changes.

We implement all methods in Java.

Datasets and Query Sets We use two real datasets and one synthetic dataset in our experiments, as follows:

1. `DMV`: This dataset contains the vehicle registration records of New York State [88]. It contains 11,944,194 rows. Here, the queries ask for the number of valid registrations for vehicles produced within a certain date range. Answering these queries involves predicates on three attributes: `model_year`, `registration_date`, and `expiration_date`. (We study a larger number of predicates in Section 5.6.)
2. `Instacart`: This dataset is the sales records of an online grocery store [87]. We use their `orders` table, which contains 3.4 million sales records. Here, the queries ask for the reorder frequency for orders made during different hours of the day. Answering these queries involves predicates on two attributes: `order_hour_of_day` and `days_since_prior`.

Dataset	**Method**	**# of Queries**	# of **Model Parameters**	**Rel. Error**	**Per-Query Time**	**Speedup**
DMV	ISOMER	150	63392	14.0 %	2105 ms	**313×**
	QuickSel	700	2800	4.68 %	6.7 ms	
Instacart	ISOMER	140	8787	8.50 %	853 ms	**178×**
	QuickSel	600	2400	7.18 %	4.8 ms	

(a) Efficiency comparison for similar errors

Dataset	**Method**	**# of Queries**	# of **Model Parameters**	**Abs. Error**	**Error Reduction**
DMV	ISOMER	60	5641	0.0360	**75.3%**
	QuickSel	700	2800	0.0089	
Instacart	ISOMER	60	1957	0.0047	**46.8%**
	QuickSel	700	2800	0.0026	

(b) Accuracy comparison for similar training time

Table 3: Summary of the comparison between the most accurate existing technique for selectivity estimation (i.e., ISOMER) and ours (i.e., QuickSel). See Figure 3 for the detailed results.

3. `Gaussian`: We also generated a synthetic dataset using a bivariate dimensional normal distribution. We varied this dataset to study our method under workload shifts, different degrees of correlation between the attributes, and more. Here, the queries count the number of points that lie within a randomly generated rectangle.

For each dataset, we measured the estimation quality using 100 test queries not used for training.

Environment All our experiments were performed on `m5.4xlarge` EC2 instances, with 16-core Intel Xeon 2.5GHz and 64 GB of memory running Ubuntu 16.04.

Metrics All reported errors are relative errors:

$$\text{Rel. Error (\%)} = \frac{1}{t}\sum_{i=1}^{t} \frac{\text{abs(true\_sel} - \text{est.\_sel)}}{\max(\text{true\_sel},\ \epsilon)} \times 100\%$$

Similar to previous work [86], here the max operator in the denominator is to guard against zero or extremely small selectivity values appear (we used ϵ=0.001).

When reporting training time, we include the time required for refining a model using an additional observed query, which itself includes the time to store the query and run the necessary optimization routines.

5.2 End-to-End Estimation Quality

In this section, we compare the end-to-end selectivity estimation quality of QuickSel versus query-driven histograms. We have summarized the main results in Table 3. The table reports that, for both `DMV` and `Instacart` dataset, QuickSel was significantly faster for the same accuracy, and significantly more accurate for the same time limit.

Specifically, we gradually increased the number of observed queries provided to each method from 10 to 1,000. For each number of observed queries, we measured the estimation error and training time of each method using 100 test queries. These results are reported in Figure 3. Given the same number of observed queries, QuickSel's training was significantly faster (Figures 3a and 3d) while still achieving comparable estimation errors (Figures 3b and 3e). We also studied the relationship between errors and training times in Figures 3c and 3f, confirming QuickSel's superior efficiency (STHoles, ISOMER+QP, and QueryModel are omitted in these figures due to their poor performance). In summary, QuickSel was able to quickly learn from a large number of observed queries (i.e., shorter training time) and produce highly accurate models.

5.3 Comparison to Scan-based Methods

We also compared QuickSel to two automatically-updating scan-based methods, AutoHist and AutoSample, which incorporate SQL Server's automatic updating rule into equi-width multidimensional histograms and samples, respectively. Since both methods incur an up-front cost for obtaining their statistics, they should produce relatively more accurate estimates initially (before seeing new queries). In contrast, the accuracy of QuickSel's estimates should quickly improve as new queries are observed.

To verify this empirically, we first generated a `Gaussian` dataset (1 million tuples) with correlation 0. We then inserted 200K new tuples generated from a distribution with a *different* correlation after processing 200 queries, and repeated this process. In other words, after processing the first

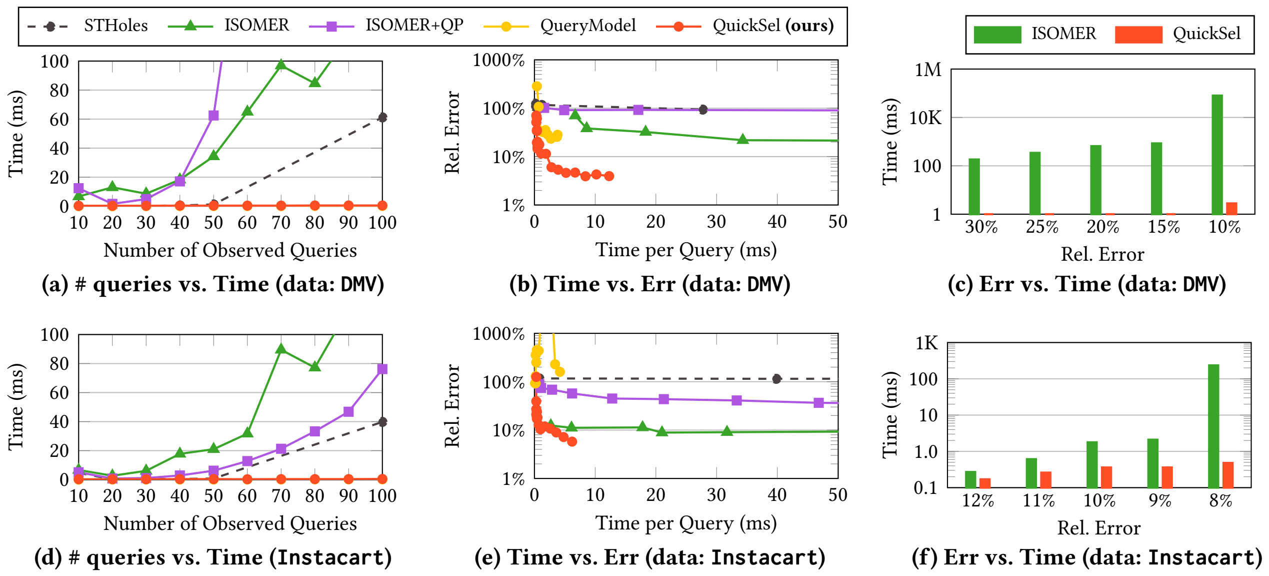


Figure 3: Comparison between QuickSel and state-of-the-art query-driven histograms. (Left) QuickSel's per-query runtime was extremely low. (Middle) QuickSel was the most accurate for the same time budget *(note: QueryModel's error reached 8,793% for* Instacart*).* (Right) QuickSel required significantly less time for the same accuracy.

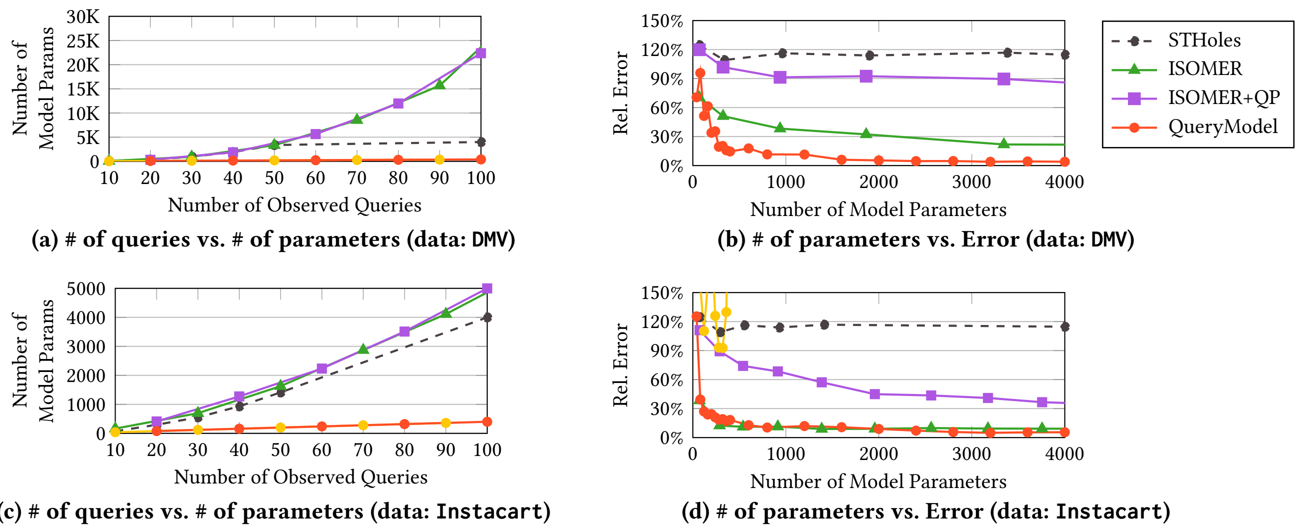


Figure 4: Comparison between QuickSel's model and the models of query-driven histograms. (Left) For the same number of observed queries, QuickSel created the least number of model parameters. (Right) QuickSel's model was more effective in expressing the data distribution, yielding the least error.

100 queries, we inserted new data with correlation 0.1; after processing the next 100 queries, we inserted new data with correlation 0.2; and continued this process until a total of 1000 queries were processed. We performed this process for each method under comparison. QuickSel adjusted its model each time after observing 100 queries. AutoHist and AutoSample updated their statistics after each batch of data insertion. QuickSel and AutoHist both used 100 parameters (# of subpopulations for the mixture model and # of buckets for histograms); AutoSample used a sample of 100 tuples.

Figure 5a shows the error of each method. As expected, AutoHist produced more accurate estimates initially. However, as more queries were processed, the error of QuickSel drastically decreased. In contrast, the errors of AutoSample

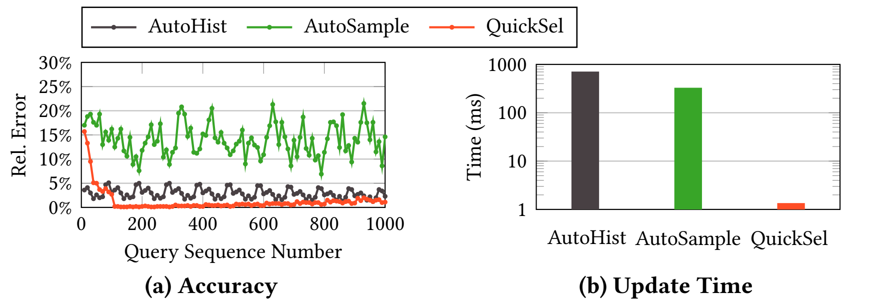


Figure 5: QuickSel versus periodically updating scan-based methods (given the same storage size).

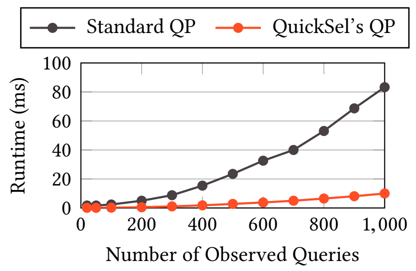


Figure 6: QuickSel versus Standard quadratic programming (QP).

and AutoHist did not improve with more queries, as they only depend on the frequency at which a new scan (or sampling) is performed. After processing only 100 queries (i.e., initial update), QuickSel produced more accurate estimates than both AutoHist and AutoSample. On average (including the first 100 queries), QuickSel was 57.3% and 91.1% more accurate than AutoHist and AutoSample, respectively. This is consistent with the previously reported observations that query-driven methods yield better accuracy than scan-based ones [21]. (The reason why query-driven proposals have not been widely adopted to date is due to their prohibitive cost; see Section 7.2). In addition, Figure 5b compares the update times of the three methods. By avoiding scans, QuickSel's query-driven updates were 525× and 243× faster than AutoHist and AutoSample, respectively.

5.4 QuickSel's Optimization Efficiency

To study the QuickSel's optimization efficiency, we compared two approaches for solving the quadratic problem defined in Theorem 1: solving the original QP without any modifications versus solving our modified version (Problem 3). We used the `cvxopt` library for the former and used `jblas` (a linear algebra library) for the latter. Both libraries use multiple cores for parallel processing.

Figure 6 shows the time taken by each optimization approach. The second approach (Problem 3) was increasingly more efficient as the number of observed queries grew. For example, it was 8.36× faster when the number of observed queries reached 1,000. This is thanks to the modified problem having an analytical solution, while the original problem required an iterative gradient descent solution.

5.5 QuickSel's Model Effectiveness

In this section, we compare the effectiveness of QuickSel's model to that of models used in the previous work. Specifically, the effectiveness is assessed by (1) how the model size—its number of parameters—grows as the number of observed queries grows and (2) how quickly its error decreases as its number of parameters grows. Here, we use QuickSel's default setting whereby its number of parameters increases linearly with the number of observed queries, deferring the analysis of its non-default setting to Section 5.6.

Figures 4a and 4c report the relationship between the number of observed queries and the number of model parameters. As discussed in Section 2.3, the number of buckets (hence, parameters) of ISOMER increased quickly as the number of observed queries grew. STHoles was able to keep the number of its parameters small due to its bucket merging technique; however, this had a negative impact on its accuracy. Here, QuickSel used the least number of model parameters. For instance, when 100 queries were observed for `DMV`, QuickSel had 10× fewer parameters than STHoles and 56× fewer parameters than ISOMER.

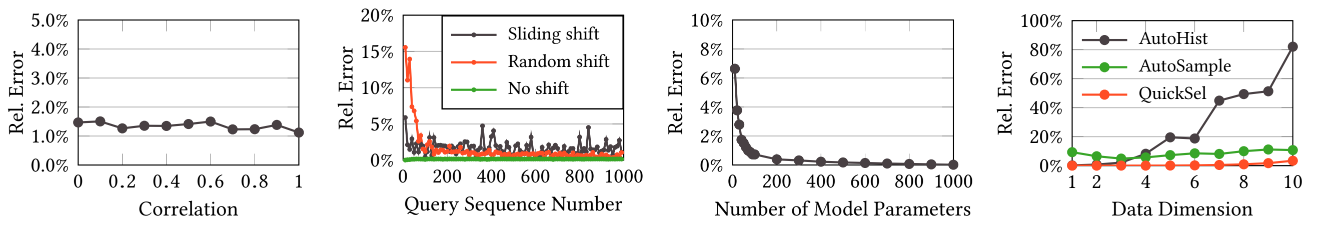
We also studied the relationship between the number of model parameters and the error. The lower the error (for the same number of model parameters), the more effective the model. Figures 4b and 4d show the result. Given the same number of model parameters, QuickSel produced significantly more accurate estimates. Equivalently, QuickSel produced the same quality estimates with much fewer model parameters.

5.6 Robustness

We further studied how QuickSel's accuracy is affected by data distribution, query workloads, number of model parameters, and number of columns in the schema.

Data Correlation To study how QuickSel's accuracy changes based on different degrees of correlations in the data, we used our `Gaussian` workload, generating values with different correlations between columns. In each case, QuickSel trained its model using 100 observed queries; the error was measured using the other 100 queries (not used for training). As shown in Figure 7a, the errors remained almost identical across all different degrees of correlation.

Workload Shifts We also studied QuickSel's accuracy under situations where the query workload shifts over time.



(a) Data Correlation **(b) Workload Shift** **(c) Model Param Count** **(d) Data Dimension**

Figure 7: Impact of data correlation, workload shifts, the number of model parameters, and data dimension on QuickSel's accuracy. The 2-dim `Gaussian` dataset was used except for in (d) where the dataset dimension was varied.

Here, we generated the `Gaussian` dataset with correlation 0.5. We generated queries with different predicates, with each predicate being a different rectangle in the 2-dimensional space.[10] We simulated three scenarios of workload shifts by modifying these rectangular predicates. First, we created a *random-shift* workload by choosing random rectangles in the space. We also created a *sliding-shift* workload by gradually moving the rectangles from the left-tail of the normal distribution towards the right-tail. Third, we created a *no-shift* workload by using the same rectangle for all queries.

In each scenario, we first trained QuickSel on the first 10 observed queries (i.e., sequence numbers: 1–10) and measured the accuracy on the next 10 observed queries (i.e., sequence numbers: 11–20). Then, we trained QuickSel on the first 20 observed queries (i.e., sequence numbers: 1–20) and measured the accuracy using the next 10 observed queries (i.e., their sequence numbers: 21–30). We kept increasing the number of observed queries until we reached 1,000 observed queries for training.

The results are plotted in Figure 7b. As expected, the errors were highest under random-shift. However, the error still decreased as QuickSel observed more queries; after 100 queries, the relative error was only 1.2% for the random shift-workload. The errors were lower in general for the other workloads.

Model Parameter Count To study the relationship between QuickSel's number of parameters and its accuracy, we disabled QuickSel's default setting (i.e., # of model params = 4 × # of observed queries). Instead, we manually controlled its number of parameters.

Figure 7c shows the results for a `Gaussian` dataset with correlation 0.5. As expected, the errors were relatively higher when the number of model parameters was extremely small (i.e., 10). However, as soon as the number of model parameters reached 50, the errors were drastically reduced.

[10]The purpose of this scenario is to simulate a workload shift; we study high-dimensional data and complex queries in Figure 7d.

Data Dimension Lastly, we studied the effect of data dimension (i.e., number of columns) on error. Here, we generated the datasets using multivariate normal distributions with different dimensions. For each dataset, we used three methods—AutoHist, AutoSample, and QuickSel—to produce selectivity estimates for the predicates on all dimensions. AutoHist used 1000 buckets, AutoSample used 1000 sampled rows, and QuickSel used 1000 observed queries.

Figure 7d shows the result. The error of AutoHist increased quickly as the dimensions increased, which is a well-known problem of multidimensional histograms. In contrast, the errors of AutoSample and QuickSel were not as sensitive to the number of dimensions of the data. The robustness of QuickSel is due to the fact that its internal model only depends on the intersection sizes of the query predicates (not on their dimensions). Among these methods, QuickSel was the most accurate.

6 INTEGRATION WITH EXISTING DBMS

In this paper, we study the query-driven selectivity estimation (Problem 1) as a standalone problem, which is not tied to any specific DBMS. However, any query-driven selectivity estimation technique (including ours) can be integrated into a DBMS using much of their existing infrastructure. Most DBMS systems contain the module that computes actual selectivities, the module that computes selectivity estimates, and the API to store metadata in its system catalog. For example, Apache Spark already collects actual selectivities in the `FilterExec` class (defined in `basicPhysicalOperators.scala`) [1]. Although Spark currently reports this selectivity to the user only at query time, it be modified to also store the observed selectivities in its metastore (which is equivalent to its system catalog). The produced selectivity estimates can then be used in the `FilterEstimation` class (defined in `FilterEstimation.scala`). The previous work proposes a similar integration strategy but for IBM DB2 [89] and Microsoft SQL Server [14].

Table 4: Comparison of selectivity estimation methods

Approach	Model	Method	Key Contributions
Based on Database Scans (Scan-based Selectivity Estimation)	Scan-based Histograms	Lynch [66]	Introduces multidimensional histograms
		Muralikrishna [74]	Introduces equidepth histograms
		Van Gelder [97]	Estimates Join selectivity with histograms for *important* domains
		GOH [43]	Optimizes single-attribute histograms for joint distribution
		MHIST [28]	Partitions columns for scalable multidimensional histograms
		Thaper [94]	Builds histograms over streaming data
		CORDS [41]	Identifies correlated columns for building multidimensional histograms
		To [95]	Builds histograms with entropy as a metric
	Sampling	Lipton [65]	Introduces adaptive sampling for high accuracy
		Haas [36]	Uses sampling for join selectivity estimation
		Riondato [81]	Guarantees accuracy relying on the VC-dimension of queries
	KDE	GenHist [33, 34]	Applies kernel density estimation to selectivity estimation
		Heimel [39]	Proposes an optimal bandwidth adjustments for KDE
Based on Observed Queries (Query-driven Selectivity Estimation)	Error-feedback Histograms *(fast but less accurate)*	ST-histogram [13]	Refines the bucket frequencies based on the errors
		LEO [89]	Identifies incorrect statistics using observed queries
		STHoles [21]	Proposes a new buckets split mechanism; adopted by ISOMER
		SASH [63]	Proposes a *junction tree* model for finding the best set of histograms
		QueryModel [15]	Avoids modeling the data distribution by using queries directly
	Max-Entropy Histograms *(accurate but slow)*	ISOMER [69, 70, 86]	Finds a maximum entropy distribution consistent with observed queries
		Kaushik et al. [51]	Extends ISOMER for distinct values
		Ré et al. [79, 80]	Seeks the max entropy distribution based on *possible worlds*
	Mixture Model ***(fast & accurate)***	**QuickSel (Ours)**	Employs a mixture model for selectivity estimation; develops an efficient training algorithm for the new model

7 RELATED WORK

There is an extensive body of work on selectivity estimation due to its importance for query optimization. In this section, we review both the scan-based methods (Section 7.1) as well as the query-driven ones (Section 7.2). QuickSel belongs to the latter category. We also discuss the connection to self-tuning/self-driving databases. Finally, we briefly overview the selectivity estimation in commercial DBMS (Section 7.3). We have summarized the related work in Table 4.

7.1 Database Scan-based Estimation

As explained in Section 1, we use the term *scan-based methods* to refer to techniques that directly inspect the data (or part of it) for collecting their statistics. These approaches differs from query-based methods which rely only on the actual selectivities of the observed queries.

Scan-based Histograms These approaches approximate the joint distribution by periodically scanning the data. There has been much work on how to efficiently express the joint distribution of multidimensional data [24, 26, 28, 32–34, 36, 38, 39, 41–43, 48, 50, 61, 65, 66, 72, 74, 81, 94, 95, 97]. There is also some work on histograms for special types of data, such as XML [12, 18, 98, 100], spatial data [47, 55–57, 64, 68, 75, 90, 92, 93, 99, 102, 103], graph [29], string [44–46, 71]; or for privacy [37, 60, 62].

Sampling Sampling-based methods rely on a sample of data for estimating its joint distribution [36, 65, 81]. However, drawing a new random sample requires a table-scan or random retrieval of tuples, both of which are costly operations and hence, are only performed periodically.

Kernel Density Estimation (KDE) KDE is a technique that translates randomly sampled data points into a distribution [85]. In the context of selectivity estimation, KDE has been used as an alternative to histograms [33, 34, 39]. The similarity between KDE and mixture models (which we employ for QuickSel) is that they both express a probability density function as a summation of some basis functions. However, KDE and MM (mixture models) are fundamentally different. KDE relies on independent and identically distributed samples, and hence lends itself to scan-based selectivity estimation. In contrast, MM does not require any sampling and can thus be used in query-driven selectivity estimation (where sampling is not practical).

7.2 Query-driven Estimation

Query-driven techniques create their histogram buckets adaptively according to the queries they observe in the workload. These techniques can be further categorized into two based on how they compute their bucket frequencies: error-feedback histograms and max-entropy histograms.

Error-feedback Histograms Error-feedback histograms [13, 15, 21, 53, 54, 63] adjust bucket frequencies in consideration of the errors made by old bucket frequencies. They differ in how they create histogram buckets according to the observed queries. For example, STHoles [21] splits existing buckets with the predicate range of the new query. SASH [63] uses a space-efficient multidimensional histogram, called MHIST [28], but determines its bucket frequencies with an error-feedback mechanism. QueryModel [15] treats the observed queries themselves as conceptual histogram buckets and determines the distances among those buckets based on the similarities among the queries' predicates.

Max-Entropy Histograms Max-entropy histograms [51, 69, 70, 80, 86] find a maximum entropy distribution consistent with the observed queries. Unfortunately, these methods generally suffer from the exponential growth in their number of buckets as the number of observed queries grows (as discussed in Section 2). QuickSel avoids this problem by relying on mixture models.

Fitting Parametric Functions Adaptive selectivity estimation [23] fits a parametric function (e.g., linear, polynomial) to the observed queries. This approach is more applicable when we know the data distribution *a priori*, which is not assumed by QuickSel.

Self-tuning Databases Query-driven histograms have also been investigated in the context of self-tuning databases. IBM's LEO [89] corrects errors in any stage of query execution based on the observed queries. Microsoft's AutoAdmin [14, 22] focuses on automatic physical design, self-tuning histograms, and monitoring infrastructure. ST-histogram [13] and STHoles [21] (see Table 4) are part of this effort. DBL [76] and IDEA [30] exploit the answers to past queries for more accurate approximate query processing. QueryBot 5000 [67] forecasts the future queries, whereas OtterTune [96] and index [58] use machine learning for automatic physical design and building secondary indices, respectively.

7.3 Estimation in Commercial DBMS

Selectivity estimation in Oracle 12c supports both sampling [77] and scan-based histograms [5]. Oracle 12c also reuses the selectivity of an observed query if the *same query* is issued again [4]. PostgreSQL [9], IBM [3], MariaDB [6], and SQL Server [11] all rely on scan-based histograms for selectivity estimation. In particular, SQL Server offers an option for automatic histogram updates whereby histograms are updated when more than a certain percentage of the data has been updated (20% by default). Apache Hive [82] and Apache Spark [40] use cost-based optimizers based on the ranges of attribute values and the number of distinct values in each column. This approach can be regarded as a simple form of histograms where each bucket is a distinct value and bucket frequencies are uniform.

8 CONCLUSION AND FUTURE WORK

The prohibitive cost of query-driven selectivity estimation techniques has greatly limited their adoption by DBMS vendors, which for the most part still rely on scan-based histograms and samples that are periodically updated and are otherwise stale. In this paper, we proposed a new framework, called *selectivity learning* or QuickSel, which learns from every query to continuously refine its internal model of the underlying data, and therefore produce increasingly more accurate selectivity estimates over time. QuickSel differs from previous query-driven selectivity estimation techniques by (i) not using histograms and (ii) enabling extremely fast refinements using its mixture model. We formally showed that the training cost of our mixture model can be reduced from exponential to only quadratic complexity (Theorem 1). We also conducted an extensive empirical study with various datasets and workloads, confirming that QuickSel can achieve the same accuracy as state-of-the-art query-driven histograms but 254.6× faster on average. Further, given the same space budget, QuickSel produced on average 57.3% and 91.1% more accurate estimates than periodically-updated histograms and samples, respectively.

We plan to investigate several related problems:

Impact of Selectivity Learning on Query Optimization We have demonstrated QuickSel's superior accuracy and performance for selectivity estimation; however, we have not studied its impact on query optimization (including access paths and join planning). We plan to integrate QuickSel into an open-source DBMS in order to study its impact on the overall quality of the query plans chosen by the query optimizer.

Join Selectivity Learning Accurate join selectivity estimation requires the correlation information between the join keys. We plan to extend QuickSel to incorporate these statistics in its mixture model.

REFERENCES

[1] https://github.com/apache/spark/blob/master/sql/core/src/main/

scala/org/apache/spark/sql/execution/basicPhysicalOperators.scala. [Online; accessed September-16-2018].

[2] Apache commons: Optimization. http://commons.apache.org/proper/commons-math/userguide/optimization.html. [Online; accessed September-16-2018].

[3] Collecting histogram statistics. https://www.ibm.com/support/knowledgecenter/SSEPEK_11.0.0/perf/src/tpc/db2z_collecthistogramstatistics.html. [Online; accessed September-16-2018].

[4] Database sql tuning guide: Automatic reoptimization. https://docs.oracle.com/database/121/TGSQL/tgsql_optcncpt.htm#TGSQL227. [Online; accessed September-16-2018].

[5] Database sql tuning guide: Histograms. https://docs.oracle.com/database/121/TGSQL/tgsql_histo.htm#TGSQL95249. [Online; accessed September-16-2018].

[6] Histogram-based statistics. https://mariadb.com/kb/en/library/histogram-based-statistics/. [Online; accessed September-16-2018].

[7] Joptimizer. http://www.joptimizer.com/. [Online; accessed September-16-2018].

[8] MATLAB: quadprog. https://www.mathworks.com/help/optim/ug/quadprog.html/. [Online; accessed September-16-2018].

[9] Postgresql 9.2.24 documentation. https://www.postgresql.org/docs/9.2/static/row-estimation-examples.html. [Online; accessed September-16-2018].

[10] Statistical maintenance functionality (autostats) in sql server. https://support.microsoft.com/en-us/help/195565/statistical-maintenance-functionality-autostats-in-sql-server. [Online; accessed September-16-2018].

[11] Statistics. https://docs.microsoft.com/en-us/sql/relational-databases/statistics/statistics?view=sql-server-2017. [Online; accessed September-16-2018].

[12] A. Aboulnaga, A. R. Alameldeen, and J. F. Naughton. Estimating the selectivity of xml path expressions for internet scale applications. In *VLDB*, 2001.

[13] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at data. *SIGMOD*, 1999.

[14] S. Agrawal, N. Bruno, S. Chaudhuri, and V. R. Narasayya. Autoadmin: Self-tuning database systemstechnology. *IEEE Data Eng. Bull.*, 2006.

[15] C. Anagnostopoulos and P. Triantafillou. Learning to accurately count with query-driven predictive analytics. In *Big Data*, 2015.

[16] M. Andersen, J. Dahl, and L. Vandenberghe. Cvxopt: A python package for convex optimization. 2013.

[17] T. Asparouhov and B. Muthén. Structural equation models and mixture models with continuous nonnormal skewed distributions. *Structural Equation Modeling: A Multidisciplinary Journal*, 2016.

[18] S. S. Bhowmick, E. Leonardi, and H. Sun. Efficient evaluation of high-selective xml twig patterns with parent child edges in tree-unaware rdbms. In *SIGMOD*, 2007.

[19] C. M. Bishop. Pattern recognition. *Machine Learning*, 2006.

[20] B. Blohsfeld, D. Korus, and B. Seeger. A comparison of selectivity estimators for range queries on metric attributes. In *SIGMOD Record*, 1999.

[21] N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: a multidimensional workload-aware histogram. In *SIGMOD*, 2001.

[22] S. Chaudhuri and V. Narasayya. Self-tuning database systems: a decade of progress. In *PVLDB*, 2007.

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

[24] L. Chen and M. T. Ozsu. Multi-scale histograms for answering queries over time series data. In *ICDE*, 2004.

[25] Y.-C. Chen. Lecture 6: Density estimation: Histogram and kernel density estimator. http://faculty.washington.edu/yenchic/18W_425/Lec6_hist_KDE.pdf. [Online; accessed September-16-2018].

[26] G. Cormode, M. Garofalakis, P. J. Haas, C. Jermaine, et al. Synopses for massive data: Samples, histograms, wavelets, sketches. 2011.

[27] P. F. Craigmile and D. Tirrerington. Parameter estimation for finite mixtures of uniform distributions. *Communications in Statistics-Theory and Methods*, 1997.

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

[29] J. Feng, Q. Qian, Y. Liao, G. Li, and N. Ta. Dmt: a flexible and versatile selectivity estimation approach for graph query. In *WAIM*, 2005.

[30] A. Galakatos, A. Crotty, E. Zgraggen, C. Binnig, and T. Kraska. Revisiting reuse for approximate query processing. *PVLDB*, 2017.

[31] A. Genz. Numerical computation of multivariate normal probabilities. *Journal of computational and graphical statistics*, 1992.

[32] S. Guha, N. Koudas, and D. Srivastava. Fast algorithms for hierarchical range histogram construction. In *PODS*, 2002.

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

[34] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Selectivity estimators for multidimensional range queries over real attributes. *VLDBJ*, 2005.

[35] A. Gupta and T. Miyawaki. On a uniform mixture model. *Biometrical Journal*, 1978.

[36] P. J. Haas, J. F. Naughton, and A. N. Swami. On the relative cost of sampling for join selectivity estimation. In *PODS*, 1994.

[37] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 2010.

[38] Z. He, X. Xu, S. Deng, and B. Dong. K-histograms: An efficient clustering algorithm for categorical dataset. *arXiv preprint cs/0509033*, 2005.

[39] M. Heimel, M. Kiefer, and V. Markl. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation. In *SIGMOD*, 2015.

[40] R. Hu, Z. Wang, W. Fan, and S. Agarwal. Cost based optimizer in apache spark 2.2. https://databricks.com/blog/2017/08/31/cost-based-optimizer-in-apache-spark-2-2.html, 2018. [Online; accessed September-16-2018].

[41] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, 2004.

[42] Z. Istvan, L. Woods, and G. Alonso. Histograms as a side effect of data movement for big data. In *SIGMOD*, 2014.

[43] H. Jagadish, H. Jin, B. C. Ooi, and K.-L. Tan. Global optimization of histograms. *SIGMOD Record*, 2001.

[44] H. Jagadish, O. Kapitskaia, R. T. Ng, and D. Srivastava. Multi-dimensional substring selectivity estimation. In *VLDB*, 1999.

[45] H. Jagadish, O. Kapitskaia, R. T. Ng, and D. Srivastava. One-dimensional and multi-dimensional substring selectivity estimation. *VLDB*, 2000.

[46] H. Jagadish, R. T. Ng, and D. Srivastava. Substring selectivity estimation. In *PODS*, 1999.

[47] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, 1998.

[48] J. Jestes, K. Yi, and F. Li. Building wavelet histograms on large data in mapreduce. *PVLDB*, 2011.

[49] H. Joe. Approximations to multivariate normal rectangle probabilities based on conditional expectations. *Journal of the American Statistical Association*, 1995.

[50] P. Karras and N. Mamoulis. Lattice histograms: a resilient synopsis structure. In *ICDE*, 2008.

[51] R. Kaushik and D. Suciu. Consistent histograms in the presence of distinct value counts. *PVLDB*, 2009.

[52] M. S. Kester, M. Athanassoulis, and S. Idreos. Access path selection in main-memory optimized data systems: Should i scan or should i probe? In *SIGMOD*, 2017.

[53] A. Khachatryan, E. Müller, C. Stier, and K. Böhm. Sensitivity of self-tuning histograms: query order affecting accuracy and robustness. In *SSDBM*, 2012.

[54] A. Khachatryan, E. Müller, C. Stier, and K. Böhm. Improving accuracy and robustness of self-tuning histograms by subspace clustering. *TKDE*, 2015.

[55] G. Koloniari, Y. Petrakis, E. Pitoura, and T. Tsotsos. Query workload-aware overlay construction using histograms. In *CIKM*, 2005.

[56] F. Korn, T. Johnson, and H. Jagadish. Range selectivity estimation for continuous attributes. In *ssdbm*, 1999.

[57] N. Koudas, S. Muthukrishnan, and D. Srivastava. Optimal histograms for hierarchical range queries. In *PODS*, 2000.

[58] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *SIGMOD*, 2018.

[59] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 1951.

[60] Y.-H. Kuo, C.-C. Chiu, D. Kifer, M. Hay, and A. Machanavajjhala. Differentially private hierarchical count-of-counts histograms. *PVLDB*, 2018.

[61] E. Lam and K. Salem. Dynamic histograms for non-stationary updates. In *IDEAS*, 2005.

[62] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *PODS*, 2010.

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

[64] X. Lin, Q. Liu, Y. Yuan, and X. Zhou. Multiscale histograms: Summarizing topological relations in large spatial datasets. In *VLDB*, 2003.

[65] R. J. Lipton, J. F. Naughton, and D. A. Schneider. *Practical selectivity estimation through adaptive sampling*. 1990.

[66] C. A. Lynch. Selectivity estimation and query optimization in large databases with highly skewed distribution of column values. In *VLDB*, 1988.

[67] L. Ma, D. Van Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon. Query-based workload forecasting for self-driving database management systems. In *SIGMOD*, 2018.

[68] N. Mamoulis and D. Papadias. Selectivity estimation of complex spatial queries. In *International Symposium on Spatial and Temporal Databases*, pages 155–174, 2001.

[69] V. Markl, P. J. Haas, M. Kutsch, N. Megiddo, U. Srivastava, and T. M. Tran. Consistent selectivity estimation via maximum entropy. *VLDBJ*, 2007.

[70] V. Markl, N. Megiddo, M. Kutsch, T. M. Tran, P. Haas, and U. Srivastava. Consistently estimating the selectivity of conjuncts of predicates. In *PVLDB*, 2005.

[71] A. Mazeika, M. H. Böhlen, N. Koudas, and D. Srivastava. Estimating the selectivity of approximate string queries. *TODS*, 2007.

[72] G. Moerkotte, D. DeHaan, N. May, A. Nica, and A. Boehm. Exploiting ordered dictionaries to efficiently construct histograms with q-error guarantees in sap hana. In *SIGMOD*, 2014.

[73] G. Moser, S. H. Lee, B. J. Hayes, M. E. Goddard, N. R. Wray, and P. M. Visscher. Simultaneous discovery, estimation and prediction analysis of complex traits using a bayesian mixture model. *PLoS genetics*, 2015.

[74] M. Muralikrishna and D. J. DeWitt. Equi-depth multidimensional histograms. In *SIGMOD Record*, 1988.

[75] T. Neumann and S. Michel. Smooth interpolating histograms with error guarantees. In *British National Conference on Databases*, 2008.

[76] Y. Park, A. S. Tajik, M. Cafarella, and B. Mozafari. Database learning: Toward a database that becomes smarter every time. In *SIGMOD*, 2017.

[77] C. Proteau. Guide to performance and tuning: Query performance and sampled selectivity. http://www.oracle.com/technetwork/products/rdb/0403-sampled-selectivity-128646.pdf. [Online; accessed September-16-2018].

[78] S. Ragothaman, S. Narasimhan, M. G. Basavaraj, and R. Dewar. Unsupervised segmentation of cervical cell images using gaussian mixture model. In *CVPR Workshops*, 2016.

[79] C. Ré and D. Suciu. Understanding cardinality estimation using entropy maximization. In *PODS*, 2010.

[80] C. Ré and D. Suciu. Understanding cardinality estimation using entropy maximization. *TODS*, 2012.

[81] M. Riondato, M. Akdere, U. Çetintemel, S. B. Zdonik, and E. Upfal. The vc-dimension of sql queries and selectivity estimation through sampling. In *ECML PKDD*, 2011.

[82] J. C. RodrÃŋguez. An overview on optimization in apache hive: Past, present future. https://www.slideshare.net/HadoopSummit/an-overview-on-optimization-in-apache-hive-past-present-future. [Online; accessed September-16-2018].

[83] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 2000.

[84] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD*, 1979.

[85] B. W. Silverman. *Density estimation for statistics and data analysis*. 2018.

[86] U. Srivastava, P. J. Haas, V. Markl, M. Kutsch, and T. M. Tran. Isomer: Consistent histogram construction using query feedback. In *ICDE*, 2006.

[87] J. Stanley. 3 million instacart orders, open sourced. https://www.instacart.com/datasets/grocery-shopping-2017, 2017. [Online; accessed September-16-2018].

[88] State of New York. Vehicle, snowmobile, and boat registrations. https://catalog.data.gov/dataset/vehicle-snowmobile-and-boat-registrations, 2018. [Online; accessed September-16-2018].

[89] M. Stillger, G. M. Lohman, V. Markl, and M. Kandil. Leo-db2's learning optimizer. In *VLDB*, 2001.

[90] J. Sun, Y. Tao, D. Papadias, and G. Kollios. Spatio-temporal join selectivity. *Information Systems*, 2006.

[91] A. Swami and K. B. Schiefer. On the estimation of join result sizes. In *EDBT*, 1994.

[92] M. Tang and F. Li. Scalable histograms on large probabilistic data. In *KDD*, 2014.

[93] Y. Tao, J. Sun, and D. Papadias. Selectivity estimation for predictive spatio-temporal queries. In *ICDE*, 2003.

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

[95] H. To, K. Chiang, and C. Shahabi. Entropy-based histograms for selectivity estimation. In *CIKM*, 2013.

[96] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic database management system tuning through large-scale machine learning. In *SIGMOD*, 2017.

[97] A. Van Gelder. Multiple join size estimation by virtual domains. In *PODS*, 1993.

[98] C. Wang, S. Parthasarathy, and R. Jin. A decomposition-based probabilistic framework for estimating the selectivity of xml twig queries.

In *EDBT*, 2006.

[99] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang. Selectivity estimation on streaming spatio-textual data using local correlations. *PVLDB*, 2014.

[100] Y. Wu, J. M. Patel, and H. Jagadish. Using histograms to estimate answer sizes for xml queries. *Information Systems*, 2003.

[101] J. Yang, X. Liao, X. Yuan, P. Llull, D. J. Brady, G. Sapiro, and L. Carin. Compressive sensing by learning a gaussian mixture model from measurements. *IEEE Transactions on Image Processing*, 2015.

[102] Q. Zhang and X. Lin. On linear-spline based histograms. In *WAIM*, 2002.

[103] Q. Zhang and X. Lin. Clustering moving objects for spatio-temporal selectivity estimation. In *Australasian database conference*, 2004.

A RELATIONSHIP TO MAX ENTROPY

The max-entropy query-driven histograms optimize their parameters (i.e., bucket frequencies) by searching for the parameter values that maximize the entropy of the distribution $f(x)$. We show that this approach is approximately equivalent to QuickSel's optimization objective, i.e., minimizing MSE of $f(x)$ from a uniform distribution. The entropy of the probability density function is defined as $-\int f(x)\,\log(f(x))\,dx$. Thus, maximizing the entropy is equivalent to minimizing $\int f(x)\,\log(f(x))\,dx$, which is related to minimizing MSE as follows:

$$\begin{aligned}\arg\min \int f(x)\,\log(f(x))\,dx &\approx \arg\min \int f(x)\,(f(x)-1)\,dx \\ &= \arg\min \int (f(x))^2\,dx\end{aligned}$$

since $\int f(x)\,dx = 1$ by definition. We used the first-order Taylor expansion to approximate $\log(x)$ with $x-1$. Note that, when the constraint $\int f(x)\,dx = 1$ is considered, $f(x) = 1/|R_0|$ is the common solution to both the entropy maximization and minimizing MSE.

B ANALYSIS OF ITERATIVE SCALING

The previous work uses an algorithm called *iterative scaling* to optimize the bucket frequencies. In this section, we analyze why using the approach is non-trivial when some of buckets are merged or pruned. Specifically, we show that the approach becomes non-trivial if a histogram bucket may partially overlap with a predicate range, rather than the bucket is completely within the predicate range or is completely outside the predicate range.

Selectivity Estimation with Histograms First, we describe how histograms can be used for selectivity estimation. This description is needed to present iterative scaling itself. Let G_z for $z = 1, \dots, m$ denote the boundary of z-th bucket. w_z is the frequency of the z-th bucket. Then, the histogram approximates the distribution of data as follows:

$$f(x) = \frac{w_j}{|G_j|} \qquad \text{for } G_j \text{ such that } x \in G_j$$

For a query's predicate P_i, its selectivity can be computed as follows:

$$s_i = \sum_{j=1}^{m} \frac{|B_i \cap G_j|}{|G_j|}\,\boldsymbol{w} = \boldsymbol{s}$$

One can observe that the above expression is akin to selectivity estimation formula with a mixture model (Section 3.2), which is natural since mixture models can be regarded as a generalization of histograms.

Optimization with Maximum Entropy Principle To optimize the bucket frequencies, the previous work employs the maximum entropy principle. When the maximum entropy principle is used, one can optimize the bucket frequencies by solving the following problem.

Problem 4 (Training with Max Entropy Principle)

$$\begin{aligned}&\arg\min_{\boldsymbol{w}} \quad \int f(x)\,\log(f(x))\,dx \\ &\textit{such that} \quad A\,\boldsymbol{w} = \boldsymbol{s}\end{aligned}$$

where A is a n-by-m matrix; its (i,j)-th entry is defined as

$$(A)_{i,j} = \frac{|B_i \cap G_j|}{|G_j|}.$$

$\boldsymbol{s}$ is a size-n column vector; its i-th entry is the observed selectivity of the i-th query.

If a histogram bucket is completely included within a predicate range, $A_{i,j}$ takes 1; if a histogram bucket is completely outside a predicate range, $A_{i,j}$ takes 0. If a histogram bucket partial overlaps with a predicate range, $A_{i,j}$ takes a value between 0 and 1.

For histograms, the integral in the above problem can be directly simplified as follows:

$$\begin{aligned}\int f(x)\,\log(f(x))\,dx &= \sum_{i=1}^{m} \int_{x\in G_i} \frac{w_i}{|G_i|}\log\left(\frac{w_i}{|G_i|}\right)\,dx \\ &= \sum_{i=1}^{m} w_i\,\log\left(\frac{w_i}{|G_i|}\right)\end{aligned}$$

Iterative Scaling Iterative Scaling solves the above problem by updating model parameters in a sequential order; that is, it updates w_1 as using fixed values for other parameters, it updates w_2 as using fixed values for other parameters, and so on. This iteration (i.e., updating all w_1 through w_m) continues until those parameter values converge. In this process, the important part is the formula for the updates.

To derive this update rule, the previous work uses the Lagrangian method, as follows. In the following derivation, we suppose a slightly more general setting; that is, we allow possible partial overlaps. We first define L:

$$L(\boldsymbol{w},\boldsymbol{\lambda}) = \sum_{j=1}^{m} w_j \log\left(\frac{w_j}{|G_j|}\right) - \boldsymbol{\lambda}^\top(A\boldsymbol{w} - \boldsymbol{s})$$

where $\boldsymbol{\lambda}$ is a size-m column vector containing m Lagrangian multipliers, i.e., $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_m)^\top$.

At optimal solutions, the derivative of L with respect to $\boldsymbol{w}$ and λ must be zero, respectively. Let the column vectors of A be denoted by $a_1, \ldots, a_m$; that is, $A = [a_1, \ldots, a_m]$. Then,

$$\frac{\partial L}{\partial w_j} = \log(w_j) + 1 - \log(|G_j|) - a_j^\top \boldsymbol{\lambda} = 0$$

Let $z_i = \exp(\lambda_i)$. Then,

$$\begin{aligned}
&\log\left(\frac{w_j}{|G_j|}\right) = a_j^\top \boldsymbol{\lambda} - 1 \qquad \text{for } j = 1, \ldots, m \\
\Rightarrow\quad &\frac{w_j}{|G_j|} = \frac{1}{e} \exp\left(a_j^\top \boldsymbol{\lambda}\right) \qquad \text{for } j = 1, \ldots, m \\
\Rightarrow\quad &w_j = \frac{|G_j|}{e} \prod_{i=1}^{n} \exp\left(A_{i,j}\, \lambda_i\right) \qquad \text{for } j = 1, \ldots, m \\
\Rightarrow\quad &w_j = \frac{|G_j|}{e} \prod_{i=1}^{n} z_i^{A_{i,j}} \qquad \text{for } j = 1, \ldots, m \qquad (6)
\end{aligned}$$

From the constraint that $\sum_{j=1}^{m} A_{i,j} w_j = s_i$ for $i = 1, \ldots, n$,

$$\sum_{j=1}^{m} A_{i,j} \frac{|G_j|}{e} \prod_{i=1}^{n} z_i^{A_{i,j}} = s_i \qquad (7)$$

Let's assume that $A_{i,j}$ always takes either 0 or 1 (the condition used in the previous work); then, the above expression can be simplified to produce an analytic update rule. Let C_i be an index set such that $C_i = \{k \mid A_{i,k} = 1, k = 1, \ldots, m\}$. Also, let $D_{j\setminus i} = \{k \mid A_{k,j} = 1, k = 1, \ldots, n, k \neq i\}$. Then,

$$\begin{aligned}
&\sum_{j=1}^{m} A_{i,j} \frac{|G_j|}{e} \prod_{i=1}^{n} z_i^{A_{i,j}} = s_i \\
\Rightarrow\quad &\sum_{j \in C_i} \frac{|G_j|}{e}\, z_i \prod_{k \in D_{i\setminus j}} z_k = s_i \\
\Rightarrow\quad &z_i = \frac{s_i}{\sum_{j \in C_i} |G_j|/e \prod_{k \in D_{i\setminus j}} z_k} \qquad (8)
\end{aligned}$$

Using the above equation, the previous work continues to update z_i for $i = 1, \ldots, n$ until convergence. Once those values are obtained, the bucket frequencies can be obtained by Equation (6).

However, without the assumption that $A_{i,j}$ always takes either 0 or 1, obtaining the update equation (Equation (8)) from Equation (7) is non-trivial.

C PROOF

Proof to Theorem 1. The theorem can be shown by substituting the definition of QuickSel's model (Equation (2)) into the probability density function $f(x)$ in Equation (3). Note that minimizing $(f(x) - g_0(x))^2$ is equivalent to minimizing $f(x)\,(f(x) - 2\,g_0(x))$, which is also equivalent to minimizing $(f(x))^2$ since $g_0(x)$ is constant over B_0 and $\int f(x)\,dx = 1$.

The integration of $(f(x))^2$ over B_0 can be converted to a matrix multiplication, as shown below:

$$\begin{aligned}
\int (f(x))^2\, dx &= \int \left[\sum_{z=1}^{m} \frac{w_z\, I(x \in G_z)}{|G_z|}\right]^2 dx \\
&= \int \sum_{i=1}^{m}\sum_{j=1}^{m} \frac{w_i w_j}{|G_i||G_j|} I(x \in G_i)\, I(x \in G_j)\, dx \\
&= \sum_{i=1}^{m}\sum_{j=1}^{m} \frac{w_i\, w_j}{|G_i|\,|G_j|} \int I(x \in G_i \wedge x \in G_j)\, dx \\
&= \sum_{i=1}^{m}\sum_{j=1}^{m} \frac{w_i w_j}{|G_i||G_j|} |G_i \cap G_j| \\
&= \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}^\top \begin{bmatrix} \frac{|G_1 \cap G_1|}{|G_1||G_1|} & \cdots & \frac{|G_1 \cap G_m|}{|G_1||G_m|} \\ \vdots & & \vdots \\ \frac{|G_m \cap G_1|}{|G_m||G_1|} & \cdots & \frac{|G_m \cap G_m|}{|G_m||G_m|} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \\
&= \boldsymbol{w}^\top Q\, \boldsymbol{w}
\end{aligned}$$

Second, we express the equality constraints in an alternative form. Note that the left-hand side of each equality constraint, i.e., $\int_{B_i} f(x)\,dx$, can be expressed as:

$$\begin{aligned}
\int_{B_i} f(x)\, dx &= \int_{B_i} \sum_{j=1}^{m} \frac{w_j}{|G_j|} I(x \in G_j)\, dx \\
&= \sum_{j=1}^{m} \frac{w_j}{|G_j|} \int_{B_i} I(x \in G_j)\, dx \\
&= \sum_{j=1}^{m} \frac{w_j}{|G_j|} |B_i \cap R_j| \\
&= \begin{bmatrix} \frac{|B_i \cap G_1|}{|G_1|} & \cdots & \frac{|B_i \cap G_m|}{|G_m|} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}
\end{aligned}$$

Then, the equality constraints, i.e., $\int_{B_i} f(x)\,dx = s_i$ for $i = 1, \ldots, n$, can be expressed as follows:

$$\begin{aligned}
&\begin{bmatrix} \frac{|B_1 \cap G_1|}{|G_1|} & \cdots & \frac{|B_1 \cap G_m|}{|G_m|} \\ \vdots & \ddots & \vdots \\ \frac{|B_n \cap G_1|}{|G_1|} & \cdots & \frac{|B_n \cap G_m|}{|G_m|} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix} = \begin{bmatrix} s_1 \\ \vdots \\ s_m \end{bmatrix} \\
\Rightarrow\quad &A\boldsymbol{w} = \boldsymbol{s}
\end{aligned}$$

Finally, $\boldsymbol{w}^\top \mathbf{1} = 1$ if and only if $\int f(x) = 1$, and $\boldsymbol{w} \succcurlyeq \mathbf{0}$ for arbitrary G_z if and only if $\int f(x) \geq 0$. □