Estimating Join Selectivities using Bandwidth-Optimized Kernel Density Models

Martin Kiefer[1] Max Heimel[2] Sebastian Breß[1,3] Volker Markl[1,3]

[1]Technische Universität Berlin
first.lastname@tu-berlin.de

[2]Snowflake Computing
max.heimel@snowflake.net

[3]German Research Center for Artificial Intelligence (DFKI)
first.lastname@dfki.de

ABSTRACT

Accurately predicting the cardinality of intermediate plan operations is an essential part of any modern relational query optimizer. The accuracy of said estimates has a strong and direct impact on the quality of the generated plans, and incorrect estimates can have a negative impact on query performance. One of the biggest challenges in this field is to predict the result size of join operations.

Kernel Density Estimation (KDE) is a statistical method to estimate multivariate probability distributions from a data sample. Previously, we introduced a modern, self-tuning selectivity estimator for range scans based on KDE that outperforms state-of-the-art multidimensional histograms and is efficient to evaluate on graphics cards. In this paper, we extend these bandwidth-optimized KDE models to estimate the result size of single and multiple joins. In particular, we propose two approaches: (1) Building a KDE model from a sample drawn from the join result. (2) Efficiently combining the information from base table KDE models.

We evaluated our KDE-based join estimators on a variety of synthetic and real-world datasets, demonstrating that they are superior to state-of-the art join estimators based on sketching or sampling.

1. INTRODUCTION

In order to correctly predict the cost of candidate plans, the query optimizer of a relational database engine requires accurate information about the result sizes of intermediate plan operations [32]. The accuracy of these cardinality estimates has a direct impact on the quality of the generated query plans. Incorrect estimates are known to cause unexpectedly bad query performance [6, 15, 20, 23, 28]. In fact, due to the multiplicative nature of joins, errors in these estimates typically propagate exponentially through larger query plans [15]. This means that even small improvements can dramatically improve the information quality that is available to the query optimizer [15, 20].

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org.
Proceedings of the VLDB Endowment, Vol. 10, No. 13
Copyright 2017 VLDB Endowment 2150-8097/17/08.

A particular challenging problem is to accurately and consistently predict the result size of joins [34]. The typical approach for this is to combine the information from base table estimators under the assumptions of uniformity, independence, and containment [34]. However, while easy to compute, this approach can cause severe estimation errors if any of the underlying assumptions is violated [6]. Multiple authors suggested specialized methods to tackle the *Join Estimation* problem, including Sampling [8, 11, 21, 36], Graphical Models [9, 35], and Sketches [19, 30]. However, none of them managed to manifest themselves as a generally viable solution, leaving the problem as one of the major unsolved challenges in research on query optimization [22].

In prior work, we introduced *bandwidth-optimized Kernel Density Models (KDE)* as a way to compute multidimensional selectivity estimates. KDE is a data-driven, nonparametric method to estimate probability densities from a data sample [31]. We demonstrated that combining KDE with a query-driven tuning mechanism to optimally pick the so-called *bandwidth* yields an estimator that typically outperforms the accuracy of state-of-the-art multidimensional histograms. Furthermore, the estimator is easy to maintain and to parallelize on modern hardware [12]. In this paper, we significantly expand upon this work and demonstrate how to estimate the result size of queries spanning multiple equijoins and base table predicates from bandwidth-optimized KDE models. In particular, we explain how to compute estimates from both joint models and combined base table models. We present pruning methods to reduce the computational overhead and demonstrate a mechanism to automatically tune the bandwidth parameters. Based on an extensive experimental evaluation, we found that our family of estimators matches and usually outperforms the accuracy of existing state-of-the-art join estimators like correlated sampling [36] or the AGMS sketch [30]. Finally, we provide all sources and experimental scripts to allow others to reproduce and build upon our results.[1]

In the following two sections, we introduce background knowledge on the join estimation problem and bandwidth-optimized KDE models. In Section 4, we lay the theoretical foundation of our work, explaining how to estimate join selectivities from a KDE model. Section 5 introduces pruning techniques to reduce the computational overhead, and Section 6 discusses strategies to fine-tune the bandwidth parameter of these models. Finally, Section 7 presents our experimental evaluation, and Section 8 concludes the paper by summarizing our findings.

[1]The source code is available at: `goo.gl/RejjVk`.

2. THE JOIN ESTIMATION PROBLEM

Given a set of n relations $R_1, R_2, \ldots, R_n$, and a query $Q = \sigma_{c_1}(R_1) \bowtie_{\theta_1} \left(\ldots \bowtie_{\theta_{n-1}} \sigma_{c_n}(R_n)\right)$, where σ_{c_i} denotes a selection with (local) predicate c_i and $\bowtie_{\theta_i}$ a join with join predicate θ_i, our goal is to predict the fraction of tuples from the Cartesian Product $R_1 \times \ldots \times R_n$ that fall into the query's result. This *Join Estimation Problem* is one of the classic problems from query optimization research [22], and improving the quality of join estimates has a direct and measurable impact on the plan quality produced by cost-based optimizers [6, 15, 18, 20, 23, 28, 33]. We consider the important subproblem where all joins are equijoins.

2.1 Classic Join Estimation

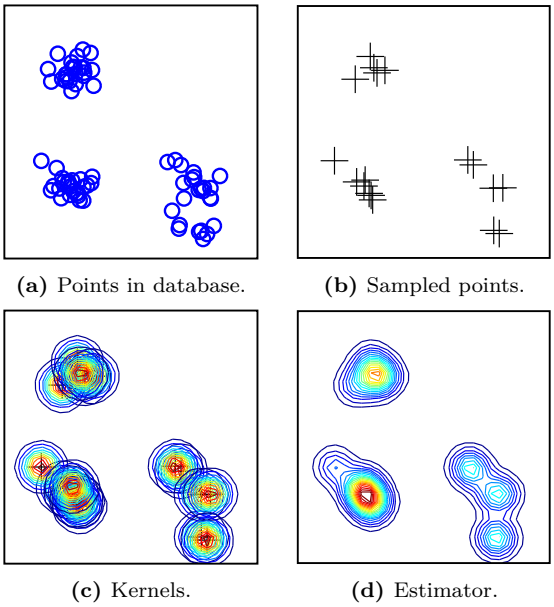
The classic way to estimate an equijoin between two tables R_1 and R_2 requires us to know the number of distinct keys $n_{R_1.A_1}$ and $n_{R_2.A_1}$ in the corresponding join columns. Assuming *uniformity*, each distinct key in R_i will appear $|R_i|/n_{R_i.A_1}$ times. Further assuming that the key domain from the table with fewer distinct keys is a subset of the other table's domain – the *containment* assumption –, and assuming that the local predicates c_1 and c_2 are *independent* of the join attribute, we arrive at the classic join estimation formula that is used by most query optimizers [32, 34]:

$$|\sigma_{c_1}(R_1) \bowtie \sigma_{c_2}(R_2)| \approx \frac{|\sigma_{c_1}(R_1)| \cdot |\sigma_{c_2}(R_2)|}{\max\left(n_{R_1.A_1}, n_{R_2.A_1}\right)} \qquad (1)$$

While straightforward to derive and easy to compute, the underlying assumptions make Equation (1) susceptible to several sources of estimation errors that can cause substantially under- or overestimations of the join result size [15, 20, 34]. This instability has inspired several researchers to investigate more sophisticated methods for estimating join result sizes. These methods can be broadly categorized into two classes: While *base table models* dynamically combine the information from individual estimators, *joint models* directly model the value distribution for preselected joins. Joint models usually produce more accurate estimates but are less flexible and harder to maintain than base table models.

2.2 Sampling-based Join Estimation

Sampling is a powerful tool to estimate selectivities for both individual and joined query results. Creating and maintaining a random sample from database tables is a well-understood topic [37], and we can directly produce estimates from base table samples by evaluating the actual query on them [11, 26]. However, joining base table samples has one major drawback: While it is an unbiased estimator to the selectivity, its variance is very high for small sample sizes due to missing join partners in the sample. While this problem can be mitigated using non-uniform methods like end-biased [7] or correlated sampling [36], the created samples are targeted to predefined joins. Another possibility is to build a joint model by sampling directly from the result of a particular join. Sampling from a join result, and maintaining said sample under updates, deletions and insertions, are well-understood problems, and can be done efficiently in the presence of join indexes [5, 26]. Such a join sample generally produces better estimates and requires smaller sample sizes compared to evaluating the query based on base table samples [11, 21]. Leis. et. al. showed that estimates computed from join samples significantly improve plan quality [21].



(a) Points in database. **(b)** Sampled points. **(c)** Kernels. **(d)** Estimator.

Figure 1: A Kernel Density Estimator approximates the underlying distribution of a given dataset (a) by picking a random sample of data points from the set (b), centering local probability distributions (kernels) around the sampled points (c), and averaging those local distributions (d).

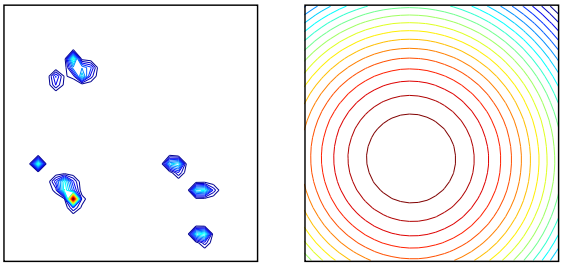
However, join samples are limited to queries that use the exact same join from which the sample was drawn.

2.3 The AGMS Sketch

The AGMS sketch is a probabilistic data structure to estimate the join size between two data streams [1]. Given a data stream $A = [v_1, v_3, v_2, \ldots]$, every distinct value v_i is associated with a uniform random variable $X_i \in \{0, 1\}$. Whenever the value v_i is encountered in stream A the counter $sk(A)$ is incremented by the value of X_i. Given a sketch $sk(B)$ constructed from the stream B with the same set of random variables, an estimate for the join cardinality $|A \bowtie B|$ is given by $sk(A) \cdot sk(B)$. While a single set of AGMS sketches does not provide reliable results, the variance of the estimator reduces by a factor n when the estimates provided by n pairs of sketches are averaged. The AGMS sketch can be extended to multiple joins. It also extends to selections by representing them as a join with all values matching the selection predicate [36].

3. BANDWIDTH-OPTIMIZED KDE

Kernel Density Estimation (KDE) is a data-driven, non-parametric method to estimate a probability density function from a data sample [31]. We illustrate the principle idea behind a KDE-based estimator in Figure 1: Based on a sample (Figure 1(b)) drawn from a table (Figure 1(a)), KDE places local probability density functions, the so-called kernels, around the sample points (Figure 1(c)). The probability density function for the overall data is then estimated by summing and averaging over those kernels (Figure 1(d)). Formally, based on a data sample $\mathcal{S} = \left\{\vec{t}^{(1)}, \ldots, \vec{t}^{(s)}\right\}$ of



(a) Bandwidth too small. **(b)** Bandwidth too large.

Figure 2: Tuning the bandwidth is crucial for the estimation quality of KDE. If the bandwidth is too small (a), the estimator overfits the sample. If it is too large (b), all local information is lost.

size s from a d-dimensional dataset, multivariate KDE defines an estimator $\hat{p}(\vec{x}) : \mathbb{R}^d \to \mathbb{R}$ that assigns a probability density to each point $\vec{x} \in \mathbb{R}^d$:

$$\begin{aligned}\hat{p}(\vec{x}) &= \frac{1}{s}\sum_{i=1}^{s} \hat{p}^{(i)}(\vec{x}) \\ &= \frac{1}{s \cdot |H|}\sum_{i=1}^{s} K\left(H^{-1}\left[\vec{t}^{(i)} - \vec{x}\right]\right) \end{aligned} \quad (2)$$

In this equation, $\hat{p}^{(i)}(\vec{x})$ denotes the local probability contribution coming from sample point $\vec{t}^{(i)}$. The function K is the *kernel function*, which defines the shape of the local probability distributions. In theory, there is a large number of possible choices for K, as any symmetric probability distribution is valid. However, in practice, the kernel function itself has only a minuscule impact on estimation quality [31], and it is typically chosen based on other desired properties. In our case, we will be using the Gaussian Kernel, a Standard Normal Distribution, since it simplifies the required derivations. The parameter H is the so-called *bandwidth matrix*, which controls the spread of the local probability distributions. Figure 2 illustrates the effects of setting this parameter for the estimator from Figure 1: If the bandwidth is chosen too small (Figure 2 (a)), the estimator is not smooth enough, resulting in a very spiky distribution that overfits the sample. On the other hand, if the bandwidth is chosen too large (Figure 2 (b)), the estimator is smoothed too strongly, losing much of the local information and underfitting the actual distribution. Accordingly, choosing the right bandwidth value is essential to achieve good estimation quality with KDE [12, 31].

In prior work [12], we demonstrated how to derive a selectivity estimator for multidimensional range queries based on KDE. We also demonstrated how to select this estimator's bandwidth parameter by numerically minimizing the estimation error. For this, we plugged the gradient of KDE's estimation error with respect to its bandwidth into an off-the-shelf numerical solver. We then continuously fed this solver with training data obtained from user queries that we collected on-the-fly. This query-driven tuning mechanism allowed us to outperform the accuracy of state-of-the-art multidimensional histograms such as GenHist [10] or STHoles [4], while still offering the flexibility and maintainability of a sample-based method. Furthermore, we explained how our estimator is efficiently evaluated and maintained on GPUs [12, 17] and, thus, identified an interesting use case for GPUs in relational databases besides actual query execution [2, 13].

4. KDE-BASED JOIN ESTIMATION

We will now discuss how to compute the result size of equijoin queries from KDE models. In particular, we introduce two basic strategies: A joint model that works by building a KDE estimator from a sample directly drawn from the join result, and a base table model that works by dynamically combining multiple base table KDE models. By combining base table models, we effectively join their estimated distributions and avoid the problem of empty join results for naive sample evaluation.

4.1 Estimating from a Join Sample

The straight-forward method to estimate joins based on KDE is to build the model from a sample that we drew directly from the join result. Sampling data directly from a join result is well-understood and can be efficiently implemented [26]. Since KDE is inherently sample-based, this method allows us to build a KDE-based join estimator without having to change a single line of code of the base table model. The advantages and disadvantages of this approach are identical to naïve sample evaluation: While we expect the joint model to produce very accurate estimates [11], it is less flexible than any method that combines base table KDE models. In fact, while the latter model can provide selectivity estimates for any arbitrary equijoin, the former requires us to construct and maintain joint models for all potential joins in the query workload.

4.2 Combining Base Table Models

Let us now derive the estimation formula for computing join estimates from individual base table models. In order to simplify this derivation, let us first consider the case of predicting the result size of a two-way equijoin query with local predicates: $Q = \sigma_{c_1}(R_1) \bowtie_{R_1.A_1 = R_2.A_1} \sigma_{c_2}(R_2)$. We will later generalize this to the case of multiple joins. For each individual join key ν, we can express the number of result tuples produced for that key as:

$$\begin{aligned}&\left|\sigma(R_1)_{R_1.A_1=\nu \wedge c_1}\right| \cdot \left|\sigma(R_2)_{R_2.A_1=\nu \wedge c_2}\right| = \\ &|R_1| \cdot p_1(R_1.A_1 = \nu \wedge c_1) \cdot |R_2| \cdot p_2(R_2.A_1 = \nu \wedge c_2)\end{aligned} \quad (3)$$

In this equation, the function $p_i(c)$ denotes the exact base table selectivity for a predicate c on table R_i. We can now define the join selectivity $J(Q) = {}^{|Q|}/_{|R_1| \cdot |R_2|}$ by summing Equation (3) over all keys $\nu \in A$, where A is the join domain, which is the set of distinct join keys. Note that A can be reduced to a subset of the distinct keys in the join columns due to local table predicates:

$$J(Q) = \sum_{\nu \in A} p_1(A_1 = \nu \wedge c_1) \cdot p_2(A_1 = \nu \wedge c_2) \quad (4)$$

We replace p_1 and p_2 by our base table estimators $\hat{p}_1$ and $\hat{p}_2$, and arrive at the join selectivity estimator $\hat{J}(Q)$:

$$\hat{J}(Q) = \sum_{\nu \in A} \hat{p}_1(A_1 = \nu \wedge c_1) \cdot \hat{p}_2(A_1 = \nu \wedge c_2) \quad (5)$$

Substituting the definition of a KDE estimator from Equation (2), we find that both estimators are evaluated and their results are multiplied. By distributivity, we can compute and multiply the individual contributions for all combinations of sample points in their respective samples S_1 and S_2, and sum over the products.

$$\begin{aligned}\hat{J}(Q) =& \frac{1}{s_1 \cdot s_2} \sum_{\nu \in A} \left(\left(\sum_{i=1}^{s_1} \hat{p}_1^{(i)} (A_1 = \nu \wedge c_1) \right) \right. \\ & \left. \left(\sum_{j=1}^{s_2} \hat{p}_2^{(j)} (A_1 = \nu \wedge c_2) \right) \right) \\ =& \frac{1}{s_1 \cdot s_2} \sum_{\nu \in A} \sum_{\substack{i=1 \\ j=1}}^{s_1, s_2} \hat{p}_1^{(i)} (A_1 = \nu \wedge c_1) \cdot \hat{p}_2^{(j)} (A_1 = \nu \wedge c_2)\end{aligned} \tag{6}$$

Assuming that the kernel functions used by our KDE models are *product kernels* [31], the following identity holds: $\hat{p}^{(i)} (A_1 = \nu \wedge c) = \hat{p}^{(i)} (A_1 = \nu) \cdot \hat{p}^{(i)} (c)$. Substituting this into Equation (6) allows us to isolate the join-specific parts of the computation:

$$\begin{aligned}\hat{J}(Q) =& \frac{1}{s_1 \cdot s_2} \sum_{\nu \in A} \sum_{\substack{i=1 \\ j=1}}^{s_1, s_2} \hat{p}_1^{(i)} (c_1) \cdot \hat{p}_1^{(i)} (A_1 = \nu) \\ & \cdot \hat{p}_2^{(j)} (c_2) \cdot \hat{p}_2^{(j)} (A_1 = \nu) \\ =& \frac{1}{s_1 \cdot s_2} \sum_{\substack{i=1 \\ j=1}}^{s_1, s_2} \hat{p}_1^{(i)} (c_1) \cdot \hat{p}_2^{(j)} (c_2) \\ & \cdot \underbrace{\left(\sum_{\nu \in A} \hat{p}_1^{(i)} (A_1 = \nu) \cdot \hat{p}_2^{(j)} (A_1 = \nu) \right)}_{\hat{J}_{i,j}}\end{aligned} \tag{7}$$

We refer to $\hat{J}_{i,j}$ as the *cross contribution*. Naively computing the cross contribution in a selectivity estimation scenario is infeasible, as the join key domain A is potentially huge and generally unknown at query optimization time. Instead, we have to exploit properties of the kernels to avoid explicitly summing over the entire join domain. Equation (8) — which is derived in Appendix A —, provides a closed-form approximation to the cross contribution for a Gaussian kernel on integer attributes. The Gaussian kernel is a common choice for KDE-based estimators and is the one used in our experimental evaluation.

$$\hat{J}_{i,j} \approx \mathcal{N}_{t_1^{(i)}, \left(\delta_1^2 + \delta_2^2\right)} \left(t_2^{(j)} \right) \tag{8}$$

In this equation, $\mathcal{N}_{\mu, \sigma^2}$ denotes the probability density function for a standard normal distribution with mean μ and variance σ^2, $t_1^{(i)}$ denotes the i-th sample point from $R_1.A_1$, and δ_1 denotes the join estimator bandwidth for R_1.

4.3 Extending to Multiple Joins

In order to generalize our approach to multiple joins, we need to introduce the notion of equivalence classes. For two tables R_i and R_j that join on the attributes $R_i.A_l$ and $R_j.A_m$, we consider the pair of attributes equivalent $R_i.A_l \sim R_j.A_m$. Note that, by definition, this equivalence also holds transitively. We denote the equivalence class for a given attribute $R_j.A_m$ by $\Psi(R_j.A_m)$. Each equivalence class contains a set of attributes that have to be equal for all tuples in the join result. Based on this definition, we can now discuss how the cross contribution can be generalized to equivalence classes containing more than two relations, and how we can compute the join selectivity for an arbitrary number of equivalence classes.

First, we consider the case of joins consisting of a single equivalence class $\Psi(R_1.A_1)$ containing n relations. Since all join attributes are in the same equivalence class, we can still sum over the shared join domain A. Assuming that each joined table R_i has a kernel density estimator model $\hat{p}_i$, we define the generalized cross contribution $\hat{J}_{o_1,\ldots,o_n}$ that needs to be computed for the cross product between all samples:

$$\hat{J}_{o_1,\ldots,o_n} = \sum_{\nu \in A} \prod_{i=1}^{k} \hat{p}_i^{(o_i)} (A_1 = \nu) \tag{9}$$

Appendix A provides a closed-form approximation for the generalized cross contribution of a Gaussian kernel. Now, since the kernels for each dimension are independent, the final formula to compute the join selectivity for n equivalence classes $\Psi_1, \ldots, \Psi_k$ over a total of n relations can be computed by multiplying their respective generalized cross contributions J_i (omitting the sample offsets for readability) with the contributions for the local predicates c_j:

$$\hat{J}(Q) = \frac{1}{\prod_{i=1}^{n} s_i} \sum_{i_1=1,\ldots,i_n=1}^{s_1,\ldots,s_n} \prod_{j=1}^{n} \hat{p}_j^{(i_j)}(c_j) \prod_{j=1}^{k} \hat{J}_j \tag{10}$$

Algorithm 1: Combining base table KDE models

```
1  # 1) Apply sample pruning:
2  S_1 = S_1 \ { t_1^(i) ∈ S_1 | p_1^(i)(c_1) < θ }
3  S_2 = S_2 \ { t_2^(i) ∈ S_2 | p_2^(i)(c_2) < θ }
4
5  # 2) Sort S_2 by the join key:
6  S_2 = sort(S_2, S_2.A_1)
7
8  # 3) Apply Cross Pruning and estimate selectivity:
9  for i in {1, ..., s_1}:
10     j = binarySearch(t_1^(i) − maxdiff(δ_1, δ_2), S_2)
11     while |t_1^(i) − t_2^(j)| ≤ maxdiff(δ_1, δ_2) ∧ j ≤ s_2:
12         Ĵ = computeĴ(t_1^(i), t_2^(j), δ_1, δ_2)
13         e += p̂_1^(i) · Ĵ · p̂_2^(j)
14     return e / (s_1 · s_2)
```

5. EFFICIENTLY JOINING KDE MODELS

In the previous section, we derived the theoretical foundation for computing join selectivities from base table models. We now discuss how we can efficiently evaluate them in practice. Our estimator receives the relational query $Q = \sigma_{c_1}(R_1) \bowtie_{R_1.A_1 = R_2.A_1} \sigma_{c_2}(R_2)$, as well as two KDE estimators with their respective base table samples S_1, S_2 and bandwidth vectors $\vec{\delta_1}, \vec{\delta_2}$. Based on Equation (7), we

know that computing the join selectivity requires us to compute the cross contributions $\hat{J}_{i,j}$ for all $s_1 \cdot s_2$ pairs from the cross product of S_1 and S_2, incurring quadratic complexity. Accordingly, a naïve implementation would severely limit the scalability and applicability of our approach. In order to reduce the number of required computations, we now introduce two pruning techniques: *Sample Pruning* and *Cross Pruning*. Algorithm 1 illustrates these methods and the general selectivity estimation procedure.

5.1 Sample Pruning

If the local contribution for a particular sample tuple is sufficiently small, the contribution of every derived tuple from the cross product will be negligible. Thus, we can omit this sample point in all following computations, which we call *Sample Pruning* (Lines 2 – 3). Similar to pushing down selections in query execution plans, we can reduce the number of input tuples that have to be considered in the more expensive computation of the cross contributions. We chose the threshold as the inverse of the cross product size $\theta = \frac{1}{r_1 \cdot r_2}$, as this limits the overall error to the join cardinality estimate to at most one tuple.

5.2 Cross Pruning

Next, we compute the cross contributions (Line 10), multiply them with their corresponding local contributions and sum them up to compute the join selectivity (Line 11). In this part of the algorithm, we apply *Cross Pruning* to reduce the computational load: As the Gaussian kernel applies smoothing by distance, its intuitive that the cross contribution for two sample points becomes negligible when the distance between the two points is very large. Again choosing the maximum tolerable error to be $\frac{1}{r_1 \cdot r_2}$, the maximum tolerable distance between two sample values is:

$$\begin{aligned} & \frac{1}{r_1 \cdot r_2} > \mathcal{N}_{t_1^i, \delta_1^2 + \delta_2^2}(t_2^j) \\ \iff & \frac{1}{r_1 \cdot r_2} > \frac{1}{\sqrt{2\pi\left(\delta_1^2 + \delta_2^2\right)}} \exp\left(-\frac{1}{2} \frac{\left(t_1^i - t_2^j\right)^2}{\left(\delta_1^2 + \delta_2^2\right)}\right) \\ \iff & \left|t_1^i - t_2^j\right| > \sqrt{-2 \cdot \ln\left(\frac{\sqrt{2\pi\left(\delta_1^2 + \delta_2^2\right)}}{r_1 \cdot r_2}\right)\left(\delta_1^2 + \delta_2^2\right)} \end{aligned} \tag{11}$$

To exploit this property, we first sort S_2 on the join attribute (Line 5). Next, instead of iterating over the cross product, the algorithm considers only tuples that are sufficiently close to each other by iterating over all tuples from S_1 (Line 7) and finding the first qualifying tuple from S_2 via binary search (Line 8). The function *maxdiff* computes the maximum distance according to Equation (11). Finally, we iterate over all qualifying tuples from S_2 (Line 9 – 11), compute the cross contribution $\hat{J}$ (Line 10) and iteratively compute the join selectivity (Line 11).

Sample pruning and cross pruning can significantly reduce the number of computations required to compute an estimate, in particular when the join and selections are very selective. Sorting, if necessary, can be done in $\mathcal{O}(s_2 \log s_2)$, pruning and computing the local contributions can be done in a single pass over each sample [12]. We have to compute s_1 binary searches, each requiring at most $\log(s_2)$ accesses to S_2. The actual number of elements traversed in the inner while-loop is data-dependent. In the degenerate case of a join that is close to a cross product, we still need to traverse S_2 for every tuple in S_1, and the complexity of the overall algorithm remains $\mathcal{O}(s_1 \cdot s_2)$. However, in the optimal case, we only have to check a handful of tuples from S_1, which yields $\mathcal{O}(s_1 log s_2)$. We argue that the degenerate case does rarely appear in real-world data and provide experimental evaluation on real-world data in Section 7.4.

5.3 Extending to Multiple Joins

Generalizing this algorithm to multiple joins requires only a few modifications. In particular, we have to apply sample pruning to all base table samples. We pick a left-deep join order and sort the samples for the right hand side of all join operators based on their join attribute. This way, we ensure that we need to sort at most $j - 1$ samples for a total of j joins. As we only allow bandwidth values such that the function values of the cross contribution never exceeds one, we can handle the joins by subsequent binary searches and apply cross pruning for each of them.

6. BANDWIDTH OPTIMIZATION

In prior work [12], we demonstrated that query-driven bandwidth optimization is crucial to the estimation quality of KDE-based selectivity estimators. During query execution, we observe the true selectivity of operators, which allows us to numerically optimize the bandwidth based on the estimation error. Since we cannot use sample or cross pruning to speed up the gradient computations for base table KDEs — a negligible contribution to the estimate does not imply a negligible contribution to the gradient —, we instead rely on a derivative-free, bound-constrained optimization algorithm. In particular, we use *constrained optimization by linear approximation* (COBYLA) [27] from nlopt [16]. We use the same algorithm for KDE over join samples.

We optimize the bandwidth based on the multiplicative error, which, given the true selectivity c and an estimate $\hat{c}$, is defined as:

$$m(c, \hat{c}) = \frac{max(c, \hat{c})}{min(c, \hat{c})} \tag{12}$$

If the estimate is larger than the actual selectivity, the multiplicative error is equivalent to the relative error, otherwise it is the inverse of the relative error. Thus, the smallest multiplicative error is 1.0 and over- and underestimations are equally penalized. The multiplicative error is the error metric of choice for cardinality estimation, as it minimizes error propagation in query plans and correlates directly with plan quality [25]. By optimizing for this error function, we ensure that the optimization translates to improved query plans.

Given a set of representative queries $Q_1, \ldots, Q_n$, we then optimize the bandwidth vectors for all tables $\vec{\delta}_1, \ldots, \vec{\delta}_m$ for the geometric mean over the estimation error:

$$\underset{\vec{\delta}_1, \ldots, \vec{\delta}_m}{\arg\min} \left(\prod_{i=0}^{n} m\left(J(Q_i), \hat{J}(Q_i)\right)\right)^{\frac{1}{n}} \tag{13}$$

Note that the estimate $\hat{J}$ depends on the bandwidth vectors $\vec{\delta}_1, \ldots, \vec{\delta}_m$. We can only optimize the bandwidth for attributes that are actually covered in the queries $Q_1, \ldots, Q_n$.

We suggest collecting query feedback for all base table filters and subsequent join operators in a query plan. This only requires keeping track of intermediate results, which can be done with very little overhead. The optimization process can then be executed periodically or triggered by a database command. Note that bandwidth optimization does not block the KDE models and, thus, optimization and estimation can be interleaved.

7. EVALUATION

In this section, we present the experimental evaluation of our KDE-based join estimators in terms of estimation quality and execution time. All experiments can be reproduced using the code and datasets from our public repository[2].

7.1 Experimental Setup

We will now describe the datasets, workloads, and estimators that were used for our experiments.

7.1.1 Compared Estimators

We compared the following estimators:

`Postgres`: Our first baseline estimator uses the EXPLAIN feature of vanilla Postgres 9.6. Postgres uses the classical join estimation formula, relying on the independence assumption and 1D statistics (frequent values, histograms, and number of distinct values).

`Table Sample (TS)`: Our second baseline estimator, which implements naïve sample evaluation based on uniform samples from the base tables.

`Join Sample (JS)`: The final baseline estimator, which implements naïve sample evaluation based on a single uniform sample from the join result.

`Correlated Sample (CS)`: A sampling-based estimator operating on biased samples constructed by using a common hash function on join attributes [36]. By correlating the samples on the join attribute, the problem of empty join results due to independence is avoided. Compared to other biased sampling algorithms, it does not require prior knowledge of the data distribution.

`AGMS`: We implemented the AGMS sketch with extensions to filter conditions as proposed in [36]. Random variables were generated by the EH3 hashing scheme which was shown to be favorable in terms of hash size and generation efficiency [29].

`JS+KDE`: Our KDE estimator based on a join sample, as described in Section 4.1.

`TS+KDE`: Our base KDE estimator based on table samples, as described in Section 5.

Note that we specifically evaluated against estimators that are comparable in terms of model construction and supported estimation operations (join subject to conjunctive base table predicates). In particular, all compared estimators can be constructed in a single pass over the data and can be maintained under updates.

All KDE models use bandwidth vectors optimized for the geometric mean of the multiplicative error on a set of 100 training queries.

[2] `https://goo.gl/RejjVk`

7.1.2 Evaluated Datasets

We conducted our experiments based on the following datasets that cover both synthetic and real-world examples:

`SN (Shifted Normal)`: Synthetic dataset consisting of 100k tuples drawn from normal distributions $\mathcal{N}_{\mu_1,\Sigma}$, $\mathcal{N}_{\mu_2,\Sigma}$, and $\mathcal{N}_{\mu_3,\Sigma}$. Values were rounded to the closest integer to simulate a discrete dataset. The covariance matrix Σ was chosen as $\left(\begin{smallmatrix}1200 & 1100\\ 1100 & 1200\end{smallmatrix}\right)$. The means were chosen as $\mu_1 = \left(\begin{smallmatrix}500 & 700\end{smallmatrix}\right)^T$, $\mu_2 = \left(\begin{smallmatrix}600 & 700\end{smallmatrix}\right)^T$ and $\mu_3 = \left(\begin{smallmatrix}700 & 700\end{smallmatrix}\right)^T$. Thus, all attributes are dense and highly correlated.

`IMDb`: Real-world dataset based on data from the Internet Movie Database[3], which we obtained using the python package IMDbPY[4] . Our queries use the tables `title` (3.5m tuples), `movie_keyword` (6m), `cast_info` (50m), `company_name` (300k), and `movie_companies` (4m).

`DMV`: Real-world dataset based on data from a Department of Motor Vehicles [14, 24]. The dataset contains information on cars, their owners and accidents in six relations containing 23 columns. The relations contain between 269 and 430k tuples.

We used dictionary encoding to transform all string attributes into integers.

7.1.3 Query Workload

Every workload in our evaluation is defined by a prepared query statement and a workload strategy (Uniform, Distinct). The prepared statement contains up to three joins as well as selections with conjunctive range and equality predicates. The left hand side of thes selection predicates is a base table attribute, while the right hand side is a parameter. We used the following algorithm to generate actual queries from the prepared statement based on the chosen workload strategy:

1. We compute the full join in the prepared statement, while ignoring the selections, and project on the non-join-attributes in the prepared statement.

2. We select a tuple t from the join result based on the specified workload strategy: (Uniform) We select a tuple from the join result with uniform probability. (Distinct) We eliminate duplicates from the join result and draw a tuple with uniform probability.

3. For attributes subject to equality predicates, the values on the previously drawn tuple t become the selection parameters. For an attribute A with range predicates, we need to provide an upper and a lower bound. We retrieve the minimum value min_A and maximum value max_A after applying the equality predicates. Defining the value of attribute A on the tuple t as $t.A$, we select the upper bound as $u_A = [T.A + (max_A - T.A) \cdot rand()]$ and the lower bound as $l_A = [T.A - (T.A - min_A) \cdot rand()]$. The function $rand$ returns a random real value in $[0, 1)$.

[3] `http://www.imdb.com/interfaces`

[4] `http://imdbpy.sourceforge.net`

The uniform strategy follows the distribution in the join result and therefore favors queries with higher selectivities. As the distinct strategy disregards the distribution of tuples in the join result, it favors queries with lower selectivities. Intuitively, a learning estimator should be more effective on the uniform workload as the queries focus on strongly represented regions. The distinct workload is harder to learn, as the estimator has to fit the entire dataset.

7.2 Estimation Quality

In the first set of experiments, we compared the accuracy of all estimators to get a feeling how our KDE-based join estimators stack up against the state of the art. For these experiments, the size of base table samples was fixed to one percent, join samples and number of AGMS sketches were chosen to match the memory required by the base table samples. While the sample sizes for correlated samples vary inherently, we chose the sampling threshold to meet the target size in expectation. The actual experiment then consisted of optimizing the bandwidth of our KDE-based models on 100 training queries, followed by measuring the multiplicative estimation error for all estimators on another 100 queries from the selected workload. We ran this experiment for different query patterns over the dataset and both workloads, repeating it 20 times for each combination.

7.2.1 SN Dataset

Figure 3 illustrates the results of this experiment for the SN dataset. SN Q1 joins the tables generated with μ_1 and μ_2 on their first attribute. The remaining two attributes are subject to range selections. Methods based on uniform base table or join samples clearly outperform the other estimators on this workload by more than 60% in terms of the median estimation error. KDE on base tables provides a small improvement of up to 15% over plain base table sample evaluation, while join samples with and without KDE both provide close to perfect estimates. Correlated samples provide the worst estimates for this query and are off by one order of magnitude. As the join is not a PK-FK join and the join attributes are skewed, the problem tackled by correlated samples does not arise.

SN Q2 adds the table generated with μ_3 to the join and introduces an additional range predicate to the query. Like for SN Q1, join sample-based estimators produce close to perfect estimates. This is not surprising, as the complexity introduced by the additional join is handled in the sampling process. TS+KDE is clearly superior to all other base table estimators and provides a better median estimation error by a factor of 4 compared to Postgres. In contrast, table samples, AGMS and correlated samples are heavily affected by the additional joins and yield estimates that are off by up to seven orders of magnitude.

7.2.2 DMV Dataset

Figure 4 illustrates the results of this experiment for the DMV dataset. We evaluate three query patterns over the four main tables of the datasets. DMV Q1 consists of a single join and four base table selections (two range predicate, two equality predicates). DMV Q2 and DMV Q3 successively add an additional join. Furthermore, they add two range selections and one equality selection, respectively.

We observe that TS+KDE is superior to the other base table estimators and outperforms them by at least one order

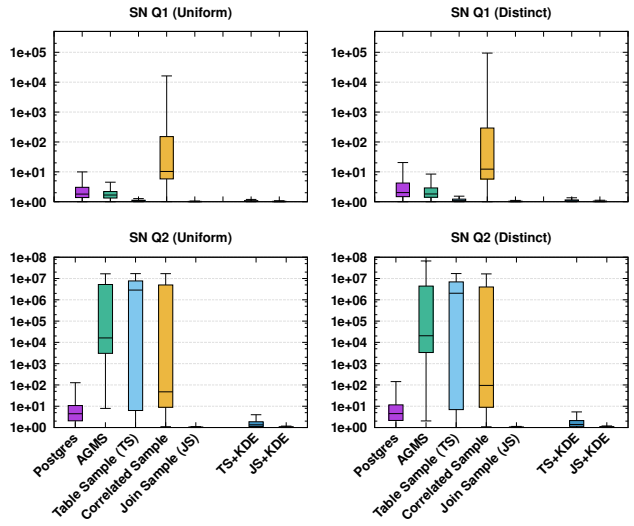


Figure 3: Estimation quality, SN dataset. The y-axis shows the multiplicative estimation error.

of magnitude in terms of the median estimation error in all experiments. Only Join Sample and JS+KDE perform better by up to a factor of four. While these estimators are very close in terms of the median estimation error, JS+KDE improves the estimation errors above the median for IMDb Q2 and Q3.

For DMV Q1, correlated sampling outperforms Table Sample by more than an order of magnitude. However, for DMV Q2 and Q3, Postgres, Table Sample and Correlated Sample perform very similar.

The AGMS sketch scales poorly with the introduced selections and performs worse than the other estimators in this experiment. Its median estimation error compared to Postgres is worse by two orders of magnitude - and the upper whisker and box boundary even extend beyond the plot boundaries. Given that the estimator variance for the AGMS sketch is proportional to the size of the cross product and selections are handled by joining with additional virtual tables [36], this behavior is expected.

7.2.3 IMDb Dataset

Figure 5 illustrates the results of this experiment for the IMDb dataset. IMDb Q1 joins two tables subject to one range and two equality predicates. IMDb Q2 and Q3 add an additional join and equality predicate respectively.

Join sample and JS+KDE provide close to perfect estimates for almost all experiments. IMDb Q3 with the distinct workload is the only exception to this: While both estimators have comparable median errors, JS+KDE shows a much better error distribution above the median as the upper box boundary and whisker improved by one and two orders of magnitude respectively.

TS+KDE provides the best estimates among the base table estimators. We see drastic improvements of an order of magnitude over correlated samples and the AGMS sketch. Compared to Table Sample, we observe drastic improvements of more than an order of magnitude for IMDBb Q2 (Distinct), Q3 (Uniform), and Q3 (Distinct) — for all other experiments the estimates are comparable. Postgres estima-

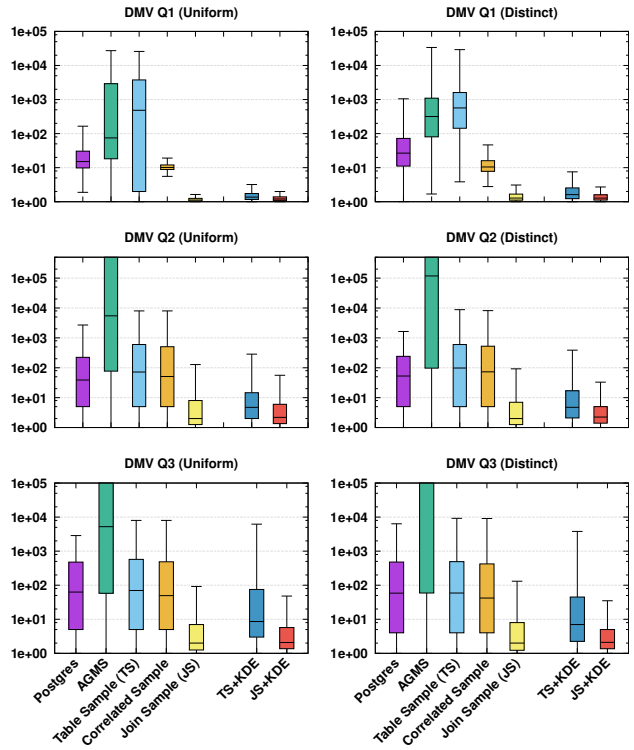


Figure 4: Estimation quality, DMV dataset. The y-axis shows the multiplicative estimation error.

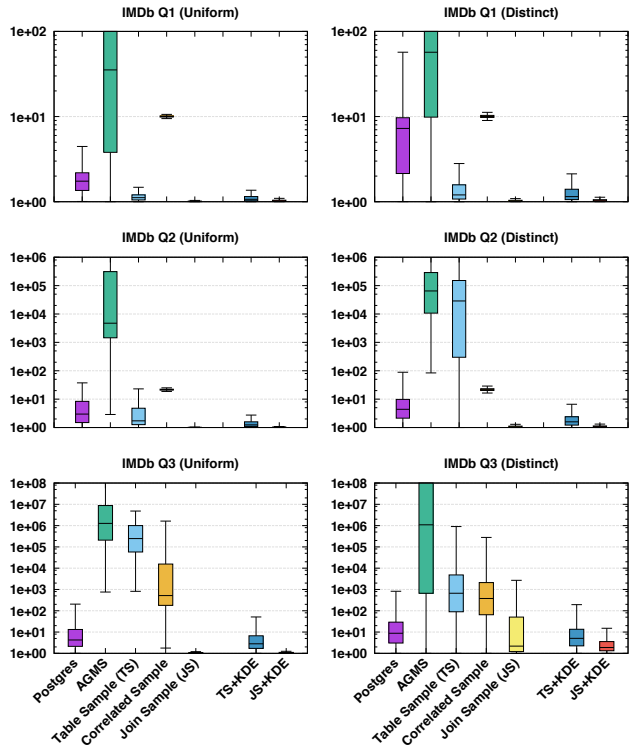


Figure 5: Estimation quality, IMDb dataset. The y-axis shows the multiplicative estimation error.

tion errors predominantly lie between one and ten, which is very competitive - especially for Q2 and Q3. However, TS+KDE still provides an improvement between factors of two and four for IMDb Q1 and Q2. For IMDb Q3, the provided estimates are comparable.

7.2.4 Discussion

Based on the results of our experiments, we can make the following general observations:

1. KDE-based join estimators generally perform better than the AGMS sketch and traditional estimators that rely on the independence assumption and 1D statistics.

2. KDE-based join estimators never perform significantly worse than their naïve sample evaluation counterparts or correlated sampling, but usually improve the estimates significantly.

7.3 Quality Impact of Model Size

In our second experiment, we investigate how the relationship between the different estimators changes with the model size. For this, we ran our previous experiments while increasing the sample ratio — and accordingly the other model sizes — by factors of two from 0.001 to 0.128. We report the geometric mean error for DMV Q1 (Uniform), as a representative for estimates on a single join, and DMV Q3 (Uniform), as a representative for multiple joins.

Figure 6 illustrate the results of this experiment for DMV Q1 with the uniform workload. There are few noteworthy observations. First, KDE-based estimators are a significant improvement over naïve sample evaluation for small sampling fractions. This is clearly visible for TS+KDE: A sampling fraction of 0.001 is not sufficient for naïve evaluation of base table samples, as the join between the two samples is likely to be empty causing estimation errors of three orders of magnitude and more. Adding a KDE model improves the estimation error by more than two orders of magnitude, which outperforms Postgres by a factor of two. While correlated samples tackle the same problem and bring substantial improvements over naïve base table samples for smaller sample sizes, KDE models are still clearly superior. Furthermore, we see JS+KDE bringing a 50% improvement over join sample evaluation for a sample size of 0.001.

Second, KDE-based estimators never perform significantly worse than Table Sample. While correlated samples bring improvements of an order of magnitude and more for small sample sizes, they converge slower to exact estimates. This causes an intersection point, after which Table Sample yields a smaller estimation error. This is consistent with our observations in Section 7.2, which showed that correlated samples are not always preferable to uniform table samples. As sample evaluation is in the parameter space of KDE-based estimators, their estimation error converges with their sample evaluation pendant for larger model sizes.

Figure 7 illustrates the results of this scaling experiment for DMV Q3 with the uniform workload, which adds two more joins to the query. The key observation is that larger sample sizes are required for the base table estimators to clearly outperform Postgres by 10% or more. TS+KDE provides better estimates at sample size 0.008, correlated samples need twice as many points. A clear improvement for table samples is only visible at a sample size of 0.128.

JS+KDE provides better estimation errors than join sample evaluation for sample sizes 0.001 to 0.004 by a factor between 1.5 and 2.

These experiments confirm that bandwidth-optimized KDE models can significantly improve the estimates computed from samples. Furthermore, the measured estimates were never significantly worse than the estimates provided by Postgres, but are usually much better depending on the sample size and the workload.

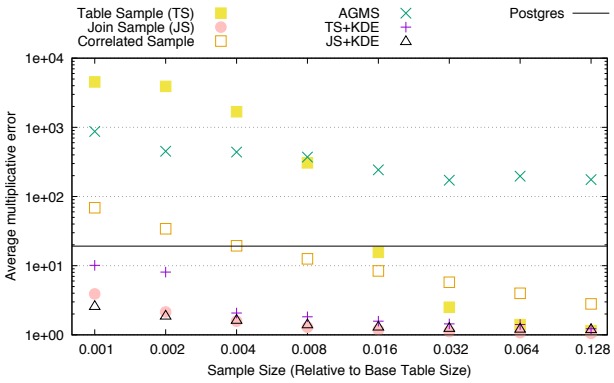


Figure 6: Estimation quality on DMV Q1 (Uniform) with growing sample sizes.

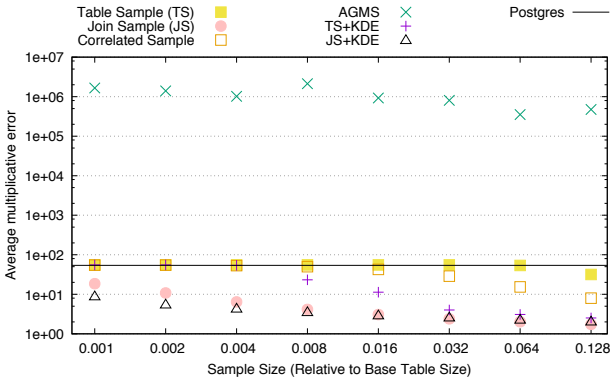


Figure 7: Estimation quality on DMV Q3 (Uniform) with growing sample sizes.

7.4 Performance Evaluation

In our final series of experiments, we evaluated the runtime scalability of our estimators for increasing sample sizes.

7.4.1 Setup

As demonstrated in our prior publication, KDE models are very well suited to be accelerated by graphics cards [12]. Accordingly, we implemented all estimators as GPU programs using custom OpenCL kernels and GPU primitives as provided by the Boost.Compute framework[5]. Naïve sample evaluation on the join sample was implemented as a straight-forward table scan, evaluation on the table samples was implemented by applying the local filter predicates on the sample, followed by performing a binary search join. Accordingly, both naïve estimators perform basically the same operations as our KDE estimators, but with less computational overhead and most aggressive pruning.

The experiment was conducted on a custom server with an AMD Opteron 6376 processor, a NVIDIA GeForce GTX 980 graphics card and 64 GB of DDR3 memory. The server was running Ubuntu Linux 16.04.1 with kernel 4.4.0. The GPU was controlled by NVIDIAs 367.57 driver.

We performed our experiments on the IMDb dataset as it is the largest of our evaluated datasets. We repeated the previous experiment for two queries over the IMDb dataset while reporting the average runtime in milliseconds instead of the estimation error.

7.4.2 IMDb Q1 (Uniform)

Figure 8 shows the results for query IMDb Q1 using the uniform workload. The first observation is that the runtime for all sample-based estimators barely increases up to a sample size of 0.016. For the AGMS sketch we observe the same effect for sample sizes of 0.001 and 0.002 This is caused by the overheads introduced by the OpenCL framework, OpenCL kernel execution and memory transfer dominating the runtime, which is 1.5ms for table sample estimators, 0.4ms for join sample estimators and 0.1ms for AGMS.

Once the actual computation dominates execution times, we see the expected linear increase in the runtime of both AGMS (0.04) and JS+KDE (0.064). As join sample evaluation only requires computationally cheap comparisons and increment operations for every tuple, the framework overhead dominates throughout the experiment.

The runtime for table sample, TS+KDE and correlated sample is very close throughout the experiment within a 10% margin. This shows the effectiveness of our pruning techniques: While computing the cross contribution is computationally expensive, our pruning techniques reduce the number of computations to a degree that they do not dominate the estimation time in this experiment.

7.4.3 IMDb Q3 (Uniform)

In order to show the performance of our estimator for multiple joins, we repeated the experiment for IMDb Q3 which adds two joins and additional base table predicates. The results are shown in Figure 9. Again, we observe that for smaller sample sizes, the framework overhead dominates the execution time. However, it is slightly larger for base table models, which is due to the additional number of involved tables. For AGMS and JS+KDE, the execution time increases roughly linear for sample sizes of 0.008 and larger. For Join Sample, the overhead dominates for all sample sizes and the execution time does not increase.

For sample sizes from 0.001 to 0.004, we see that the estimation time for TS+KDE, TS and Correlated Sample barely increase. TS+KDE adds an overhead of at most 50% to naive samples. From a sample size of 0.032 onwards, TS+KDE imposes a larger overhead and grows roughly linearly, while table and correlated samples increase sublinearly. The additional overhead imposed by KDE over Table Sample is within an order of magnitude. Only for the largest sample size of 0.128, we can observe a substantial overhead of two orders of magnitude.

Our pruning techniques are very effective for this query, as the runtime complexity without our pruning techniques would be quartic in the sample size.

[5] www.boost.org/libs/compute

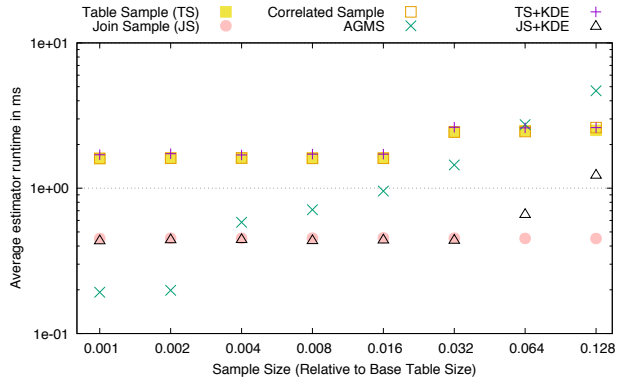


Figure 8: Estimation time on IMDb Q1 (Uniform) with growing sample sizes.

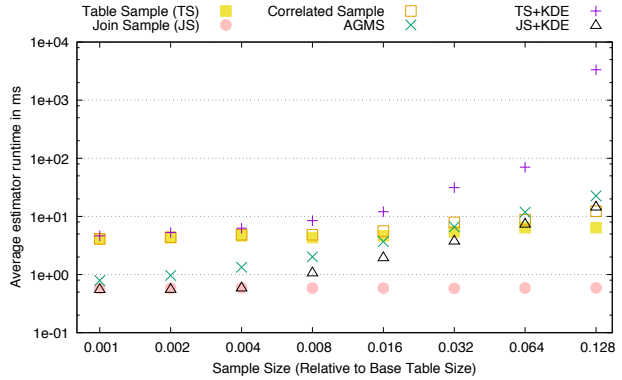


Figure 9: Estimation time on IMDb Q3 (Uniform) with growing sample sizes.

8. CONCLUSION & FUTURE WORK

In this paper, we introduced a novel way to estimate join selectivities based on bandwidth-optimized Kernel Density Estimators. Existing models suffer from at least one of the following drawbacks: They (1) provide inaccurate estimates, (2) are expensive to construct, (3) are restricted to a single type of query, or (4) are expensive or impossible to maintain under changing data.

Our approach uses KDE models, which are constructed from base table or join samples, and provides an estimate to its underlying distribution. They apply smoothing to the sample distribution by placing probability density functions on all sample points, averaging over them to compute the final estimate. The degree of smoothing is controlled by a hyper-parameter, the so-called bandwidth. Selecting this bandwidth parameter is essential for the estimation quality and can be done by performing numerical optimization over query-feedback, either based on base table or join queries. KDE combines the flexibility and maintainability of a sample-based method, with the quality of state-of-the-art selectivity estimators.

We evaluated the quality of our approach using queries on both synthetic and real-world datasets. We found that KDEs provide significantly better join estimates than traditional methods in case one of the underlying assumptions is violated. Compared to naïve sample evaluation, our models can provide significantly improved results for relatively small sample sizes, while still converging to the same accuracy for larger samples. In practice, we suggest to maintain base table KDE models for all tables in a database as they usually provide better estimates for supported operators and data types (numeric attributes, dictionary encoded attributes). Joint KDE models can be added manually, when additional accuracy is required for particular joins.

In order to conclude the paper, we now present and discuss selected topics that we believe to be interesting directions for future work in the field of KDE-based selectivity estimation:

Investigating Different Kernel Functions: As using the Gaussian kernel requires evaluating the error and exponential function, it is computationally intensive. Investigating alternative kernel functions could lead to a more efficient evaluation, while still providing similar results. In particular, one could extend our ideas to the Cauchy or Epanechnikov kernel, which are cheaper to evaluate.

Generalization to Theta Joins: Since KDE models provide a probabilistic model for the join frequency distribution of a table, they could also be used to estimate the selectivity for the more general class of theta joins. In particular, we would have to derive closed-form estimation formulae by solving Equation 14 for different integral bounds and identify how to efficiently implement those. While this is surely not trivially possible for every join predicate, we think that extending support for certain classes of general theta joins, like band joins ($R_1.L \leq R_2.A \leq R_1.U$) or simple inequality joins ($R2.x \leq R1.l$), would be possible and interesting.

KDE on runtime samples: Recent work showed promising results for uniform join samples created at estimation time in in-memory databases [21]. It would be an interesting research direction to investigate how the bandwidth of KDE models could be selected in systems without materialized samples.

Acknowledgment

The work has received funding from the European Union's Horizon2020 Research & Innovation Program under grant agreement 671500 (project 'SAGE') and from the German Ministry for Education and Research as Berlin Big Data Center BBDC (funding mark 01IS14013A).

9. REFERENCES

[1] N. Alon, P. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *PODS*, pages 10–20, 1999.

[2] S. Breß, H. Funke, and J. Teubner. Robust query processing in co-processor-accelerated databases. In *SIGMOD*, pages 1891–1906, 2016.

[3] P. Bromiley. Products and convolutions of gaussian probability density functions. 2003. `http://www.tina-vision.net`, Internal Report.

[4] N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: a multidimensional workload-aware histogram. In *SIGMOD Record*, volume 30, pages 211–222, 2001.

[5] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. In *SIGMOD*, pages 263–274, June 1999.

[6] S. Christodoulakis. Implications of certain assumptions in database performance evauation. *TODS*, 9(2):163–186, 1984.
[7] C. Estan and J. Naughton. End-biased samples for join cardinality estimation. In *ICDE*, page 20, 2006.
[8] S. Ganguly, P. Gibbons, Y. Matias, and A. Silberschatz. Bifocal sampling for skew-resistant join size estimation. In *SIGMOD Record*, volume 25, pages 271–281, 1996.
[9] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. *SIGMOD Record*, 30(2):461–472, May 2001.
[10] D. Gunopulos, G. Kollios, J. Tsotras, and C. Domeniconi. Selectivity estimators for multidimensional range queries over real attributes. *VLDB Journal*, 14(2):137–154, 2005.
[11] P. Haas, J. Naughton, and A. Swami. On the relative cost of sampling for join selectivity estimation. In *PODS*, pages 14–24, 1994.
[12] M. Heimel, M. Kiefer, and V. Markl. Self-tuning, gpu-accelerated kernel density models for multidimensional selectivity estimation. In *SIGMOD*, pages 1477–1492, 2015.
[13] M. Heimel, M. Saecker, H. Pirk, S. Manegold, and V. Markl. Hardware-oblivious parallelism for in-memory column-stores. *PVLDB*, 6(9):709–720, 2013.
[14] I. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: Automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, pages 647–658, 2004.
[15] Y. Ioannidis and S. Christodoulakis. On the propagation of errors in the size of join results. In *SIGMOD*, pages 268–277, 1991.
[16] S. G. Johnson. The NLopt nonlinear-optimization package. `http://ab-initio.mit.edu/nlopt`.
[17] M. Kiefer, M. Heimel, and V. Markl. Demonstrating transfer-efficient sample maintenance on graphics cards. In *EDBT*, pages 513–516, 2015.
[18] P.-A. Larson, W. Lehner, J. Zhou, and P. Zabback. Cardinality estimation using sample views with quality assurance. In *SIGMOD*, pages 175–186, 2007.
[19] H. Lee, R. Ng, and K. Shim. Similarity join size estimation using locality sensitive hashing. *PVLDB*, 4(6):338–349, 2011.
[20] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *PVLDB*, 9(3):204–215, 2015.
[21] V. Leis, B. Radke, A. Gubichev, A. Kemper, and T. Neumann. Cardinality estimation done right: Index-based join sampling. In *CIDR*, 2017.
[22] G. Lohman. Is query optimization a "solved" problem? SIGMOD Blog, April 2014.
[23] V. Markl, P. Haas, M. Kutsch, N. Megiddo, U. Srivastava, and T. Tran. Consistent selectivity estimation via maximum entropy. *PVLDB*, 16(1):55–76, 2007.
[24] V. Markl, V. Raman, D. Simmen, G. Lohman, H. Pirahesh, and M. Cilimdzic. Robust query processing through progressive optimization. In *SIGMOD*, pages 659–670, 2004.
[25] G. Moerkotte, T. Neumann, and G. Steidl. Preventing bad plans by bounding the impact of cardinality estimation errors. *VLDB*, 2(1):982–993, 2009.
[26] F. Olken. *Random sampling from databases*. PhD thesis, University of California at Berkeley, 1993.
[27] M. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in Optimization and Numerical Analysis*, pages 51–67, 1994.
[28] N. Reddy and J. Haritsa. Analyzing plan diagrams of database query optimizers. In *PVLDB*, pages 1228–1239, 2005.
[29] F. Rusu. *Sketches for aggregate estimations over data streams*. PhD thesis, University of Florida, 2009.
[30] F. Rusu and A. Dobra. Sketches for size of join estimation. *TODS*, 33(3):15:1–15:46, Sept. 2008.
[31] D. Scott. *Multivariate Density Estimation - Theory, Practice, and Visualization*. John Wiley & Sons, 2nd edition, 2015.
[32] P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price. Access path selection in a relational database management system. In *SIGMOD*, pages 23–34, 1979.
[33] M. Stillger, G. Lohman, V. Markl, and M. Kandil. LEO - db2's learning optimizer. In *PVLDB*, pages 19–28, 2001.
[34] A. Swami and K. Schiefer. On the estimation of join result sizes. In *EDBT*, pages 287–300, 1994.
[35] K. Tzoumas, A. Deshpande, and C. Jensen. Lightweight graphical models for selectivity estimation without independence assumptions. *VLDB*, 4(11):852–863, 2011.
[36] D. Vengerov, A. Menck, M. Zait, and S. Chakkappen. Join size estimation subject to filter conditions. *PVLDB*, 8(12):1530–1541, 2015.
[37] J. Vitter. Random sampling with a reservoir. *TOMS*, 11(1):37–57, 1985.

APPENDIX

A. GAUSSIAN CROSS CONTRIBUTION

The Gaussian kernel is a common choice for a continuous kernel in KDE. It is a Normal distribution $\mathcal{N}_{s,\delta^2}(x)$ centered on the sample point s and using the bandwidth δ as its standard deviation. In order to compute discretized probability estimates for a integer join key ν from this continuous kernel, we simply integrate it over the interval $[\nu - 0.5, \nu + 0.5]$, leaving us with the *discretized Gaussian kernel*. We assume that join attributes are not subject to selections ($A = \mathbb{Z}$) and lift this assumption in Appendix B.

A.1 Cross Contribution

Substituting the discretized Gaussian kernel into Equation 7, yields the cross contribution for the Gaussian kernel:

$$\hat{J}_{i,j} = \sum_{\nu \in \mathbb{Z}} \int_{\nu-0.5}^{\nu+0.5} \mathcal{N}_{t_1^{(i)},\delta_1^2}(x)\,dx \cdot \int_{\nu-0.5}^{\nu+0.5} \mathcal{N}_{t_2^{(j)},\delta_2^2}(x)\,dx \tag{14}$$

Equation (14) does not have a closed-form solution that would allow us to efficiently compute it without summing over all $\nu \in A$. However, for probability densities $f(x)$, $g(x)$

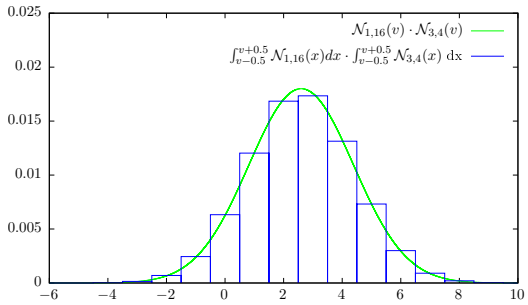


Figure 10: Approximating the cross contribution for the Gaussian kernel by the multiply-then-integrate approach

with values smaller or equal to one, we can approximate $\int_{\nu-0.5}^{\nu+0.5} f(x)\,dx \int_{\nu-0.5}^{\nu+0.5} g(x)\,dx \approx \int_{\nu-0.5}^{\nu+0.5} f(x)\,g(x)\,dx$. We illustrate this approximation in Figure 10, showing that in this case the results for first integrating then multiplying are very similar to first multiplying then integrating.

Substituting this approximation into Equation (14) gives us an approximate closed-form solution:

$$\begin{aligned}\hat{J}_{i,j} &\approx \sum_{\nu\in\mathbb{Z}} \int_{\nu-0.5}^{\nu+0.5} \mathcal{N}_{t_1^{(i)},\delta_1^2} \cdot \mathcal{N}_{t_2^{(j)},\delta_2^2} dx \\ &= \int_{-\infty}^{+\infty} \mathcal{N}_{t_1^{(i)},\delta_1^2}(x) \cdot \mathcal{N}_{t_2^{(j)},\delta_2^2}(x)\,dx \\ &= \mathcal{N}_{t_1^{(i)},(\delta_1^2+\delta_2^2)}\left(t_2^{(j)}\right) \int_{-\infty}^{+\infty} \mathcal{N}_{t',\delta'}(x) \\ &= \mathcal{N}_{t_1^{(i)},(\delta_1^2+\delta_2^2)}\left(t_2^{(j)}\right)\end{aligned} \tag{15}$$

The last transformation requires some explanation: The product $\mathcal{N}_{1,2} = \mathcal{N}_1 \cdot \mathcal{N}_2$ of two Normal probability density functions is itself a scaled Normal probability density function [3]. Its location t' and scale δ' are functions over the parameters of the individual densities. Since we integrate over the full domain, this factor integrates to one, leaving only the scaling factor, which is again given by a Normal density function that depends on the mean and bandwidth parameters of the original functions [3].

Equation (15) does not hold for densities with function values larger than one. In particular, when the bandwidth of one of the two Gaussian functions is below $(2\pi)^{-\frac{1}{2}} \approx 0.4$, the error increases drastically. However, since for this value only 1.3% of the probability mass of a Gaussian is located outside of $[\mu - 0.5, \mu + 0.5]$, the estimator is still able to fall back close to sample evaluation.

A.2 Generalized Cross Contribution

Plugging the discretized Gaussian kernel into Equation (9), we arrive at:

$$\hat{J}_{o_1,\ldots,o_n} = \sum_{\nu\in\mathbb{Z}} \prod_{i=1}^{n} \int_{\nu-0.5}^{\nu+0.5} \mathcal{N}_{t_i^{(o_i)},\delta_i^2}(x)\,dx \tag{16}$$

Similar to the single join case, we can plug in the approximation $\prod_i \int_{v-0.5}^{v+0.5} p_i(x)dx \approx \int_{v-0.5}^{v+0.5} \prod_i p_i(x)dx$. The product of k normal densities can be rewritten as a normal density and a scale factor that does not depend on the integrand [3].

$$\begin{aligned}\hat{J}_{o_1,\ldots,o_n} &= \int_{-\infty}^{\infty} \prod_{i=1}^{n} \mathcal{N}_{t_i^{(o_i)},\delta_i^2}(x)\,dx \\ &= S_{1\ldots n} \cdot \int_{-\infty}^{\infty} \mathcal{N}_{t_{1\ldots n},\delta_{1\ldots n}^2}(x)dx \\ &= S_{1\ldots n}\end{aligned} \tag{17}$$

Since we integrate over the entire domain, the normal density integrates to one, which leaves us with the scale factor $S_{1\ldots n}$ only. The scale factor $S_{1\ldots n}$ is given by:

$$S_{1\ldots n} = \frac{\sqrt{\frac{\delta_{1\ldots n}^2}{\prod_{i=1}^{n}\delta_i^2}}}{(2\pi)^{(n-1)/2}} exp\left(-\frac{1}{2}\left(\sum_{i=1}^{n}\frac{\left(t_i^{(o_i)}\right)^2}{\delta_i^2} - \frac{t_{1\ldots n}^2}{\delta_{1\ldots n}^2}\right)\right) \tag{18}$$

Finally, the quantities $\delta_{1\ldots n}^2$ and $t_{1\ldots n}$ can be computed from the individual means and variances of each normal density:

$$\begin{aligned}\delta_{1\ldots n}^2 &= \left(\sum_{i=1}^{n}\frac{1}{\delta_i^2}\right)^{-1} \\ t_{1\ldots n} &= \delta_{1\ldots n}^2 \sum_{i=1}^{n}\frac{t_i^{(o_i)}}{\delta_i^2}\end{aligned} \tag{19}$$

B. SELECTIONS ON JOIN ATTRIBUTES

The assumption that join attributes are not subject to selections can be lifted. In general, selections on the join attributes allow us to apply sample pruning for join attributes as well which potentially reduces the number of tuples that we have to consider in the cross pruning step.

Range selections require adjusting the equations for the cross contribution. If a range selection $l \le A \le u$ is applied to a join attribute, the normal density function in Equation 17 has to be considered:

$$\hat{J}_{o_1,\ldots,o_n} = S_{1\ldots n} \cdot \int_{l}^{u} \mathcal{N}_{t_{1\ldots n},\delta_{1\ldots n}^2}(x)dx \tag{20}$$

However, we do not have to adjust the maximum distance inequality (Equation 11) in cross pruning. As the additionally introduced factor $\int_l^u \mathcal{N}_{t',\delta'}(x)$ is smaller or equal than one, the given inequality remains intact even in case of range predicates on the join attribute.

Point selections are a special case of range selections. As a join attribute subject to a point selection reduces the estimate computation to multiplying estimates for a single selection per table, treating them differently simplifies the required computations: Point selections allow us to handle the joins for an entire equivalence class by local predicates. Furthermore, every table with all join attributes being subject to point selections, can even be excluded entirely from the sum over the cross product of all samples.