Investigating GPU-Accelerated Kernel Density Estimators for Join Selectivity Estimation

vorgelegt von

Martin Kiefer, B. Sc.

Matrikelnummer: 358316

dem

Fachgebiet Datenbanksysteme und Informationsmanagement
der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin

Master Thesis

5. April 2016

Gutachter:
Prof. Dr. Volker Markl

Betreuer:
Max Heimel, M. Sc.

Zeitraum:
September 2015 - März 2016

Technische Universität Berlin

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed.

Berlin, April 5, 2016 Martin Kiefer

Abstract

Kernel Density Estimators are a well-known tool from statistics to estimate probability density functions based on samples drawn from an unknown distribution. The estimate is provided by centering local probability density functions - so called *kernels* - on the sample points and averaging. The technique was successfully applied to the task of computing base table selectivities for hyper-rectangular range queries over real-valued attributes in relational databases. It was shown that estimate computations and error-driven hyper-parameter optimization can be efficiently accelerated using a GPU.

In this thesis, we extend upon this approach by using the *categorical kernel* that centers a probability mass functions on discrete valued attributes. The resulting models provide an estimate to the joint relative frequency distribution on those attributes and can be used to estimate the probability of base table selectivities subject to conjunctive equality predicates. We show how and under which assumptions independent KDE models for two different tables may be used to efficiently compute the selectivity of joins subject to such selections. We provide an algorithm to compute estimates and gradients required for hyper-parameter optimization. Furthermore, we sketch how these computations can be implemented on a GPU.

We show how our approach can be generalized to multiple subsequent joins, highlight properties of the estimator and provide experimental evaluation using an artificial and a real-world dataset.

Zusammenfassung

Kerndichteschätzer sind ein bekanntes Werkzeug aus der Statistik zur nicht-parametrischen Schätzung von Wahrscheinlichkeitsdichtefunktionen basierend auf einer Stichprobe aus einer unbekannten Verteilung. Der Schätzer wird durch das Mitteln lokaler Wahrscheinlichkeitsdichtefunktionen - sogenannten *Kernen* - erzeugt, die auf den Werten der Stichprobe zentriert werden. Kerndichteschätzer wurden erfolgreich zur Schätzung der Selektivität von hyperrechteckigen Bereichsabfragen auf Basistabellen in relationalen Datenbanken angewandt. Es wurde gezeigt, dass die Schätzungen und die fehlerbasierte Optimierung von Hyperparametern sich durch Verwendung einer GPU beschleunigen lassen.

In der folgenden Thesis erweitern wir diesen Ansatz durch Verwendung eines Kernes für kategoriale Variablen, bei dem es sich um eine Wahrscheinlichkeitsmassefunktion handelt, die auf den Werten der Stichprobe zentriert wird. Das resultierende Modell erlaubt die Schätzung der gemeinsamen Häufigkeitsverteilung der diskreten Attribute einer Tabelle. Sie erlauben die Schätzung der Selektivität von konjunktiven Gleichheitsprädikaten auf Basistabellen. Wir zeigen, wie und unter welchen Annahmen zwei unabhängige Modelle für unterschiedliche Tabellen genutzt werden können, um die Selektivitäten von Joins zwischen diesen Tabellen unter Berücksichtigung konjunktiver Gleichheitsprädikate effizient zu berechnen. Wir geben Algorithmen für die Berechnung von Schätzungen und Gradienten an, die zur fehlerbasierten Optimierung der Hyperparameter des Modells benötigt werden. Des Weiteren skizzieren wir eine mögliche Implementierung dieser Berechnungen für GPUs.

Wir zeigen, wie Berechnungen für mehrere Joins angepasst werden können, untersuchen Eigenschaften des Schätzers und führen eine Auswertung auf einem künstlich generierten und einem existieren Datensatz durch.

Contents

Abstract iii

Zusammenfassung iv

I Introduction & Problem Statement 1

1 Introduction 2

1.1 Problem Statement 4

1.2 Notation 5

1.3 Outline 6

II Background 7

2 Background 8

2.1 Selectivity Estimation in Relational Database Systems 8

2.2 Related Work 10

2.2.1 Sampling 10

2.2.2 Sketches 13

2.2.3 Graphical Models 15

2.2.4 Wavelets 16

2.3 Kernel Density Estimators 17

2.3.1 Kernel Density Estimators for Range Selectivity Estimation 19

III Approach 22

3 Approach 23

3.1 The Categorical Kernel for Discrete Attributes . . . 23

3.1.1 Using the Categorical Kernel for Selections with Equality Predicates . . . 26

3.2 Probabilistic View on Joins . . . 27

3.3 Using the Categorical Kernel for Two-Way Joins . . . 28

3.4 Algorithm . . . 32

3.4.1 Estimation . . . 32

3.4.2 Error-Driven Bandwidth Optimization . . . 36

3.4.3 GPU-Implementation . . . 40

3.4.4 Sample Maintenance . . . 43

3.5 Generalization for n-Way Joins . . . 43

3.6 Discussion . . . 45

IV Evaluation & Conclusion 48

4 Evaluation 49

4.1 Experimental Setup . . . 49

4.2 Model Size & Estimation Quality . . . 51

4.2.1 Selections . . . 51

4.2.2 Joins . . . 54

4.2.3 Interference of Selections and Join Queries in Bandwidth Optimization . . 61

4.3 Execution Time . . . 62

5 Conclusion 64

List of Tables 66

List of Figures 67

Listings 69

Bibliography 70

Part I

INTRODUCTION & PROBLEM STATEMENT

1 Introduction

Estimating the output cardinalities of operators is an important task in modern relational database systems. They are used by the query optimizer to make assumptions about the cost of possible execution plans for an incoming query [25] and therefore directly impact plan quality [40]. However, providing these estimates has to adhere to a tight time budget as it delays query execution.

Providing selectivity estimates for joins with selections on base tables is an important subproblem, as bad plan choices can significantly hurt query performance: Choosing an unfavorable join order or execution strategy can cause unnecessary large intermediate results or blocks read from disk [14]. Furthermore, estimation errors have been shown to propagate exponentially in the number of joins [25].

Contrary to the importance of the estimates, database products compute their estimates by assuming independence and uniformity of attributes [30] - assumptions that are more often violated than obeyed in real-world data and potentially cause large estimation errors. While numerous authors have proposed estimators that do not require these assumptions and have shown promising results, database systems shy away from using them due to the aforementioned performance considerations.

Independent uniform samples drawn from relations are one of the most intuitive approaches. Sample-based estimators are easy to construct [36] and to maintain [16]. They can be used to compute approximate aggregates and selectivity estimates for various operators [9] - including joins and selections. However, they need to be chosen large enough to contain enough information to provide accurate estimates. Choosing the sample large enough can be problematic in a selectivity estimation setup, as usually a fixed memory and time budget is reserved for query optimization.

For the case of hyper-rectangular range selections over continuous attributes, we have shown in a prior publication [21] that *Kernel Density Estimators* are a practical approach to selectivity estimation without the independence and uniformity assumptions. The estimator works with a sample but uses it to construct a probability density by averaging local probability density functions - so called *kernels* - centered on the sample points. The *bandwidth* is a smoothing parameter that controls the spread of the local probability densities. The estimate is then computed by integrating over the query region. Using Kernel Density Estimators has the benefits of samples when it comes to construction and adaption of the model while providing high quality estimates [21]. We have described how to perform error-driven bandwidth optimization for the model and have shown that a GPU can be utilized to evaluate larger models in the same time budget.

In this thesis, we extend our previous approach by Kernel Density Estimator models for discrete attributes. We do so by placing a probability mass function, the *categorical kernel* [31], on sample points and averaging. For a single attribute, the kernel distributes probability mass on the value of the sample point and evenly distributes the remaining probability mass across all other possible values in the domain. The amount of probability mass placed on the sample value is determined by the *bandwidth*. For multiple attributes, instances of the one dimensional kernel are multiplied. The resulting estimator is a hybrid between sampling and assuming uniformity . Like sampling, we are able to take dependencies between attributes into account, and like with the uniformity assumption, there is a general estimate for all values that are not or only partially included in the sample. The bandwidth parameter can be seen as the amount of trust that is placed in either of the approaches. Error-driven optimization of this parameter aims at balancing between the approaches to get the most out of both of them. The model proposed in this thesis is complementary to our previous approach and can be used to create models consisting of discrete and continuous attributes.

This thesis makes the following contributions:

1. We show how the categorical kernel can be used to compute the selectivity of base-table selections with equality predicates and provide the gradient that is necessary to perform error-driven bandwidth optimization.
2. We show how and under which assumption the categorical kernel can be used to efficiently estimate the selectivity of two-way equi-joins and provide the gradient for error-driven hyper-parameter optimization.
3. We provide an algorithm that allows the efficient computation of two-way join selectivity estimates by only two passes over each sample - given that the sample is sorted on the join attribute. We sketch how the algorithm can be implemented on a GPU and how it can be extended to an arbitrary number of joins.
4. We evaluate our approach on a synthetic and a real-world dataset and compare it to other estimators.

1.1 Problem Statement

We attempt to solve the selectivity estimation problem for selections with conjunctive equality predicates on base tables and joins subject to such selections.

Given a selection $\sigma_e(R)$ with a predicate e on a relation $R(A_1, ...A_{d_1})$, the selectivity $p\left(\sigma_e\left(R\right)\right)$ is computed by:

$$p\left(\sigma_e\left(R\right)\right) = \frac{\left|\sigma_e\left(R\right)\right|}{|R|} \tag{1.1}$$

We assume the selection predicate e to be the conjunction of equality predicates on the attributes of R. We represent this predicate as a set of equality predicates $(A_k = v_k)$ with $v_k \in A_k$.

$$\sigma_e\left(R\right) = \sigma_{\bigwedge_{(A_k=v_k)\in e} R.A_k=v_k}\left(R\right) \tag{1.2}$$

The task of a selectivity estimator $\hat{p}\left(\sigma_e\left(R\right)\right)$ is to provide an estimate for the true selectivity $p\left(\sigma_e\left(R\right)\right)$ such that holds:

$$\hat{p}\left(\sigma_e\left(R\right)\right)) \approx p\left(\sigma_e\left(R\right)\right) \tag{1.3}$$

Most of this thesis, we will consider queries $q(R_1, R_2)$ consisting of a single equi-join between two relations $R_1(A_1, ...A_{d_1})$ and $R_2(A_1, ...A_{d_2})$:

$$q\left(R_1, R_2\right) = \sigma_{e_1}\left(R_1\right) \bowtie_{R_1.A_j=R_2.A_i} \sigma_{e_2}\left(R_2\right) \tag{1.4}$$

The selection predicates e_1 and e_2 are conjunctive equality predicates on R_1 and R_2 respectively. We denote the true selectivity of the query by $p\left(q\left(R_1, R_2\right)\right)$:

$$p\left(q\left(R_1, R_2\right)\right) = \frac{q\left(R_1, R_2\right)}{|R_1| \cdot |R_2|} \tag{1.5}$$

Note that join selectivities without selections on base tables are just a subcase of these queries, as p_1 and p_2 may be chosen empty which lets the selection predicate always evaluate to true and disables the selection.

The task of a selectivity estimator is $\hat{p}\left(q\left(R_1, R_2\right)\right)$ is to provide estimates for the true selectivity that fulfills:

$$\hat{p}\left(q\left(R_1, R_2\right)\right) \approx p\left(q\left(R_1, R_2\right)\right) \tag{1.6}$$

We will also discuss the generalization of the problem to an arbitrary number of joins.

Evaluating and comparing the estimation quality requires an error function that quantizes the error between an estimated selectivity x and the true selectivity y. For this thesis, we choose the quadratic error $\mathcal{L}_{\text{Quadratic}}(x, y) = (x - y)^2$, mainly because the smoothing parameter of our estimator is optimized for this error. As this thesis is the first evaluation of this estimator, we choose to analyze its properties without the additional complexity of different error functions for optimization and evaluation.

Besides the quality of an estimator, we have to take the overhead into account it causes the database system to use it. There are several aspects that have to be considered.

Construction Cost: We have to consider the effort required to construct the model that the estimates are based on e.g. drawing a sample or computing frequency distributions. For small relations it might be acceptable to perform multiple passes over the data, while even one pass can be too much for very large relations. In case retrieving the data does not clearly dominate model construction, the actual cost of the model construction algorithm has to be taken into account as well.

Estimation Cost: We have to consider the cost of computing an estimate. As query optimization delays query execution this should only take a small fraction of the overall query processing time.

Memory Cost: The memory required to store the model should be reasonable and upper bounded.

Maintenance Cost: The cost required to adjust the model to changes in the database should be acceptable.

We have extracted the following requirements for a good selectivity estimator: Besides a reasonable estimation quality, the estimator should at most require one pass over the input data. The upper memory bound should be a tunable parameter that allows balancing the estimation quality for estimation cost. Maintaining the model should not require a full re-construction.

1.2 Notation

We give relational queries formally in terms of relational algebra, as it is the most concise way to express them. Note, that we always consider relations as *bags of tuples* as it is done in relational database systems [14]. Thus, base tables and intermediate results can contain duplicates.

We use the terms *two-way join*, *three-way join* and *n-way* join, to briefly refer to a join query accessing two, three and n base tables, respectively.

When we refer to a relation R, we denote its cardinality as $|R|$. An attribute A in R is referred to $R.A$ whenever ambiguities exist. When an attributes A is used in our derivations, it denotes the set of distinct values in the attribute. Thus, $|A|$ denotes the number of distinct values in attribute A.

R, R_1, R_2	Relations
S, S_1, S_2	Samples
$R.A_j$	Attribute j in relation R (Set of distinct values)
$S.A_j$	Attribute j in sample S (Set of distinct values)
q, q_j	Relational queries
$\vec{s}$	Vector/Tuple s
s_j	j-th component of vector $\vec{s}$

Table 1.1: Notation

Table 1.1 gives a quick summary of our notation throughout the paper.

We use the term *sample* for brevity throughout the thesis, in particular in context of Kernel Density Estimators for selectivity estimation. If not stated otherwise, we refer to simple random samples without replacement that are drawn independently from each other.

1.3 Outline

In Part II, we provide background on this thesis. We discuss different approaches to join selectivity estimation, formally introduce Kernel Density Estimators and explain how they can be used for selectivity estimation of hyper-rectangular range-queries on real-valued attributes.

Part III introduces our approach to selectivity estimation for joins and selections on discrete attributes. We introduce the *categorical kernel* for discrete attributes and show how estimates and gradients are computed for selections and two-way joins. We provide algorithms performing the computations, sketch a possible GPU-adaption and discuss extensions to multiple joins.

In Part IV, we provide an experimental evaluation of our approach with respect to estimation quality and execution time and, finally, summarize our findings to conclude this thesis.

Part II

Background

2 Background

In this Chapter we provide background on this thesis. In Section 2.1, we introduce the text-book approach to selectivity estimation and highlight the assumptions that are the foundation of selectivity estimation in modern relational database systems. Section 2.2 discusses related techniques that are capable of providing selectivity estimates for joins. In Section 2.3, we finally give an overview on Kernel Density Estimators. Furthermore, we explain our previous approach for selectivity estimation of hyper-rectangular range queries over real-valued attributes using Kernel Density Estimators, which is the foundation for our approach to join selectivity estimation introduced in the following Chapter.

2.1 Selectivity Estimation in Relational Database Systems

In relational database systems, SQL queries are translated into logical query plans [14], which describe the derivation of the query results in terms of operators from the relational algebra. As logical operators can be implemented by different execution strategies, and may be executable in different orders, logical query plans can be translated into a huge number of equivalent physical execution plans. A cost-based optimizer is used to find a physical execution plan that minimizes the estimated cost for executing the query [14]. For this purpose, the optimizer needs accurate estimates on the cardinalities of intermediate results. These cardinalities are estimated in a process called *selectivity estimation*, which uses data statistics to predict result sizes for the operators in the plan. The term *selectivity* is used when the result size is as a share of the maximum output cardinality of the operators. This is usually the case for selections, where the selectivities are a share of the input cardinality, or joins, where selectivities are a share of the cardinality of the cross product between input relations. As query optimization delays query execution, selectivity estimation has to adhere to a tight time budget.

Database literature [14] and introduction lectures usually cover selectivity estimation based on the number of distinct values of attributes. While the concept dates back to the early days of query optimization in the 1970s [45], it is a vital part of modern relational database systems to a large extent.

The estimates are computed based on four major assumptions [14]:

Attribute Uniformity: The values in an attribute are uniformly distributed. Thus, every value appears equally often.

Attribute Independence: The attributes in relations are pairwise statistically independent.

Containment of Value Sets: The join attributes in two relations have a subset relationship. The assumption holds for a foreign-key relationship.

Preservation of Value Sets: The number of distinct values on a non-join attribute remains the same after the join.

Given a selection with an equality predicate on an attribute A in relation R, the relation cardinality $|R|$, and the number of distinct values in attribute A denoted as $|R.A|$, the selectivity $\hat{p}\left(\sigma_{A=c}\left(R\right)\right)$ under the attribute uniformity is calculated by:

$$\hat{p}\left(\sigma_{A=c}\left(R\right)\right) = \frac{1}{|R.A|} \tag{2.1}$$

If the selection predicate is a set e of conjunctive equality predicates $(A_i = v_i)$ with $v_i \in A_i$, the estimate under the attribute independence assumption is given by multiplying the per-attribute estimates:

$$\hat{p}\left(\sigma_{e}\left(R\right)\right) = \prod_{(A_i=v_i)\in e} \hat{p}\left(\sigma_{A_i=v_i}\left(R\right)\right) \tag{2.2}$$

The selectivity of a two-way equi-join on two tables R_1 and R_2 on a common attribute A_1 given attribute uniformity assumption and containment of value sets is calculated as:

$$\hat{p}\left(R_1 \bowtie_{R_1.A_1=R_2.A_1} R_2\right) = \frac{1}{max(|R_1.A_1|, |R_2.A_1|)} \tag{2.3}$$

Without loss of generality assume $|R_1.A_1| \leq |R_2.A_1|$. The containment of value sets assumption implies that every value in $R_1.A_1$ remains in the join result. The probability for a match with a tuple from $R_2.A_1$ is then given by the attribute uniformity assumption. Thus, for every tuple from the cross product of R_1 and R_2 the join keys coincide with probability $\frac{1}{|R_2.A_1|}$.

Since the number of distinct values in the former join attribute of the join result is $|R_1.A_1|$ by containment of value sets, and preservation of value sets yields that non-join attribute frequencies are not changed, the selectivities for n-way joins can be calculated by repeated application of the two-way join formula in Equation 2.3. Using the attribute independence assumption allows to compute the estimate subject to selections by multiplication.

While the assumptions allow the efficient computation of estimates, they are often violated in real-world data sets and therefore introduce large errors. These errors have been shown to increase exponentially in the number of joins [25]. Database products dropped the attribute uniformity assumption by collecting more sophisticated per-attribute statistics like histograms and the frequencies of most common values, while maintaining the attribute independence assumption and value set assumptions. This improves the quality on selections on a single-attribute, but the remaining assumptions may still introduce large errors whenever multiple attributes are involved in selections or in joins. In a recent publication, Leis et al. have confirmed that this is a significant problem [30]: In all five evaluated relational database systems errors propagate exponentially in the number of joins and the errors are shown to cause suboptimal query plans.

Getting rid of the independence assumption means modeling the joint distribution of attributes. While numerous authors have proposed such models, they did not have product impact. We claim that this is the case because the models do either not provide reliable estimates or are too expensive to evaluate, construct, or adapt to changes in the modeled relations. In the following Section, we will introduce several models that are capable of providing estimates for joins and joins subject to selections.

2.2 Related Work

Since the introduction of the first cost-based query optimizer in 1979 [45] numerous authors have proposed selectivity estimators using various models with different theoretical background. Therefore, giving a complete overview over the landscape of selectivity estimators is infeasible. We concentrate on introducing the most prominent concepts for estimating the selectivity of equi-joins that can be used to overcome the attribute independence and attribute uniformity assumptions.

2.2.1 Sampling

Sampling-based estimators compute their estimates based on a subset of tuples from the input relations. The estimators follow a common scheme: The samples are retrieved, the join is evaluated on the samples and the estimated result is normalized to provide a valid selectivity. The construction of the sample usually involves randomness in form of random number generators or seeds for hash functions. Note that while most of the mentioned sampling estimators were introduced for two-way joins, an extension to more joins or additional selections is usually evident.

Sampling-based estimators can be distinguished by the way the sample is retrieved. The first family of algorithms creates samples by drawing portions from relations with uniform probability. This is either done on a tuple or block level. An overview of algorithms of this family is given in [18]. We will refer to the most intuitive way as *simple sampling*: Given two relations R_1 and R_2, we draw two samples S_1 and S_2 of desired size $|S_1|$ and $|S_2|$ uniformly and independently without replacement from the respective relations. Given a query $q\left(R_1, R_2\right) = \sigma_{e_1}\left(R_1\right) \bowtie_j \sigma_{e_2}\left(R_2\right)$ with selection predicates e_1 and e_2 as well as the equi-join predicate j the estimate $\hat{p}\left(q\left(R_1, R_2\right)\right)$ can be computed by the following formula:

$$\hat{p}\left(q\left(R_1, R_2\right)\right) = \frac{1}{|S_1| \cdot |S_2|} |q\left(S_1, S_2\right)| \tag{2.4}$$

Samples can be likewise used for multiple joins by executing it on the samples and normalizing with the cardinality of the cross product. It can be shown that this estimator is unbiased, which means that the expected value of the estimator is equal to the estimated quantity:

$$\mathbb{E}\left[\hat{p}\left(q\left(R_1, R_2\right)\right)\right] = p\left(q\left(R_1, R_2\right)\right) \tag{2.5}$$

Note that, even though the estimator is unbiased, the join result of two simple samples does not yield a uniform sample of the actual join result. Using independent simple samples is very convenient: They may also be used to compute other selectivity estimates and approximate aggregates [9]. In particular, they may also be used to produce selectivity estimates for per-table selections by applying the query to the sample and normalizing with the sample size. Their construction can be realized with $O(|S|)$ disk accesses [36], but at worst a sequential scan over the input relation suffices if *Reservoir Sampling* [52] is used. Maintaining a random sample for a relation that is subject to insertions, deletions and updates is a well-researched topic [16]. However, using simple sample for joins suffers from a drawback: When the number of join keys is very large compared to the sample size and there is no significant skew in the data, the probability that a join key is included in both samples is very low, which will cause the estimator to return a selectivity of zero most of the time. This is a common case in databases, as joins often involve attributes being unique keys due to database normalization. The problem deteriorates with the number of joins [9].

Adaptive Sampling Algorithms (e.g. [18, 32, 33]) use more complex sampling schemes that increase the sample size dynamically to give limits on the estimation error with a specified confidence. However, the use of adaptive sampling for selectivity estimation in a database system is of limited use: As usually a fixed time budget is reserved for selectivity estimation, the algorithm would have to abort after the sample size has reached a predefined limit and the algorithm has not reached its convergence criteria.

The second family of estimators tries to reduce the shortcomings of independent sampling by taking into account the actual contents of samples or relations during the sampling step.

The *Bifocal Sampling Algorithm* [13] tries to improve estimates by using different sampling techniques to account for sparse and dense join keys. The estimator can guarantee an error of two with high probability if the sample sizes and a threshold are selected accordingly, but requires indexes on join attributes.

The *End-Biased Sampling Algorithm* [11] does perform the sampling decisions in the join relations dependently. The algorithm uses a hash function mapping the join keys uniformly to $[0, 1]$ and a threshold parameter K_i for each of the relations R_1 and R_2. Furthermore, it requires the frequency distribution of join keys in the relations R_i denoted as $F_i(v)$. Every join key that occurs more often in R_i than K_i is automatically included in the respective sample S_i. All other join keys v are included if their hash value is less or equal than $h(v) \leq \frac{F_i(v)}{K_i}$. As the same hash function is used for both relations, all join keys v with $h(v) \leq min\left(\frac{F_1(v)}{K_1}, \frac{F_2(v)}{K_2}\right)$ are included in both samples. The estimate $\hat{p}\left(q\left(R_1, R_2\right)\right)$ for the query q is then computed as the sum of contributions c_v for every value v present *in both samples*

$$\hat{p}\left(q\left(R_1, R_2\right)\right) = \frac{1}{|R_1| \cdot |R_2|} \sum_v c_v \tag{2.6}$$

where c_v is given by [51]:

$$c_v = \begin{cases} F_1^c(v) \cdot F_2^c(v) & \text{if } F_1(v) \geq K_1 \wedge F_2(v) \geq K_2 \\ K_1 \cdot F_2^c(v) & \text{if } F_1(v) < K_1 \wedge F_2(v) \geq K_2 \\ F_1^c(v) \cdot K_2 & \text{if } F_1(v) \geq K_1 \wedge F_2(v) < K_2 \\ F_1^c(v) \cdot F_2^c(v) \cdot max\left(\frac{K_1}{F_1(v)}, \frac{K_2}{F_2(v)}\right) & \text{if } F_1(v) < K_1 \wedge F_2(v) < K_2 \end{cases} \tag{2.7}$$

The quantity $F_i^c(v)$ denotes the number of tuples in R_i with join key v that have also fulfilled the predicates of the selection. It can be shown that this is an unbiased estimator. Selecting the sampled join keys by a common hash function does address the problem of independent samples and provides a better variance of the estimate [51]. The sampling algorithm needs two passes over both relations - one for the frequency distribution and one to obtain the sample. Furthermore, the thresholds K_1 and K_2 are parameters that need to be hand-tuned as the resulting sample size is very dependent on the dataset. Note that End-Biased samples need to be constructed for every attribute that participates in a join.

The *Correlated Sampling Algorithm* [51] basically removes the guarantee that the most-common values in the relation are included in the sample but also the need to perform two passes over the relations. It requires a hash function h that maps the join keys uniformly to $[0, 1]$ as well as two target sample sizes n_1 and n_2. The target sample sizes are then used to compute the threshold $T_i = \frac{n_i}{|R_i|}$. A join key v from R_i is included if it holds $h(v) \leq T_i$. The estimate for the query q is given by:

$$\hat{p}\left(q\left(R_1, R_2\right)\right) = \frac{1}{|R_1| \cdot |R_2| \cdot min(T_1, T_2)} |q\left(S_1, S_2\right)| \tag{2.8}$$

The estimator can be shown to be unbiased. There exists a generalization for arbitrary many joins [51]. While the parameters n_1 and n_2 are referred to as target sample sizes, they only control the expected share of join keys included from the respective samples. If there is skew on the join attributes, the obtained sample sizes may vary significantly.

The term *Correlated Sampling* is also used in the *CS2* algorithm [53] for selectivity estimation of multiple joins with arbitrary join predicates. This algorithm starts at a node in the join graph and draws a simple sample. The samples for subsequent nodes in the join graph are obtained by executing the join with the simple sample and collecting the participating tuples. However, implementing the algorithm either requires an unpredictable amount of space for the samples or requires multiple passes over the data - both infeasible for selectivity estimation in database systems.

2.2.2 Sketches

Sketches are a technique from the field of stream processing. They are randomized algorithms that allow approximate answers to aggregate queries where storing the entire data stream is impossible [43]. This is done by using random seeds and the data to create random variables with distributions connected to the value of the true aggregate. Measuring characteristics of the distributions allows to derive the estimate.

The most prominent sketch for join selectivities is the *AGMS Sketch*, initially called the *Tug-Of-War Sketch* [1, 2]. Given a data stream that consists of values in $V = \{v_1, ..., v_n\}$, the sketch requires 2-wise independent random variables ξ_i for every $v_i \in V$ that have the value $+1$ or -1 with equal probability. The random variables can be obtained from a 2-wise independent hash function that is initialized with a random seed. Given a stream of values $A = [v_1, v_3, v_2, v_3, ...]$, the sketch $sk(A)$ for the data stream is given by summing up the random variables for the observed values:

$$sk(A) = \xi_1 + \xi_3 + \xi_2 + \xi_3 + ... \tag{2.9}$$

If we want to compute the estimate for a join with a second data stream B, we multiply with its sketch $sk(B)$ constructed using *the same random variables*.

$$\hat{p}\left(A \bowtie B\right) = \frac{sk\left(A\right) \cdot sk\left(B\right)}{|A| \cdot |B|} \tag{2.10}$$

It can be shown that this is an unbiased estimate for the selectivity of $A \bowtie B$. The intuition behind this estimator is, that multiplying the sketches is equivalent to multiplying every random variable in sketch A with every random variable in sketch B and summing them up. This is one summand for every tuple in the cross product of the two input relations. By 2-wise independence, the expected value of these summands is $\mathbb{E}\left[\xi_i \cdot \xi_j\right] = 1$ if $i = j$, otherwise $\mathbb{E}\left[\xi_i \cdot \xi_j\right] = 0$. Thus, in expectation we have a contribution of one for every tuple in the cross product with coinciding join values, which finally adds up to the result cardinality of the join.

The variance of the estimator reduces by a factor $\frac{1}{k}$ when the results of k sketches constructed with different random variables is averaged. Hence, usually a vector of AGMS Sketches is maintained for every stream. Note that maintaining large vectors can be computationally expensive, as every entry has to be updated for an incoming value on the stream. The variance can be further reduced by using a 4-wise independent hash function, which is more demanding in terms of the size of the random seed and computation cost [42].

The AGMS Sketch is easily extended to multiple joins by introducing further families of random variables for the additional join [10]. Every sketch is updated by adding the product of the random variables for all joins accessing this table. Consider the case of two streams $A = [v_1, v_3, v_2, v_2, v_1, ...]$ and $C = [u_1, u_2, u_1, u_3, u_2, ...]$ and an intermediate stream $B = [(v_2, u_1), (v_3, u_4), (v_1, u_2), ...]$ as well as two families of random variables ξ^1_v and ξ^2_u. The sketches $sk(A)$ and $sk(C)$ are computed like before with ξ^1_v and ξ^2_u respectively. The sketch B is computed as follows:

$$sk(B) = \xi^1_{v_2} \cdot \xi^2_{u_1} + \xi^1_{v_3} \cdot \xi^2_{u_4} + \xi^1_{v_1} \cdot \xi^2_{u_2} \tag{2.11}$$

The selectivity $\hat{p}\left(A \bowtie B \bowtie C\right)$ is then given by:

$$\hat{p}\left(A \bowtie B \bowtie C\right) = \frac{sk\left(A\right) \cdot sk\left(B\right) \cdot sk\left(C\right)}{|A| \cdot |B| \cdot |C|} \tag{2.12}$$

AGMS Sketches can even process deletions in the stream by simply decrementing the sketch by the former contribution of the deleted value.

Vengerov et. al. [51] have proposed an extension to the AGMS Sketch that allows estimating the join selectivity subject to arbitrary selection predicates using the same sketch. The authors make use of the fact that every selection on an attribute can be realized as an equi-join on this attribute with an artificial relation that contains all values satisfying the selection. Depending on the predicate of the selection, the generation of the artificial relations and the corresponding sketch can be expensive. However, in the case of equality predicates, we need to compute the hash function only once to construct the sketch for the artificial table, and in case of selections over a contiguous range summable hash functions can be used [42].

Further variants of the AGMS Sketch exist. *Fast-AGMS Sketches* [9] reduce the update time for AGMS Sketches by removing the need to update the entire AGMS sketch vector for every item in the stream. The authors propose maintaining multiple vectors of AGMS Sketches, each coming with its own family of random variables and a 2-wise independent hash function, mapping the join values to one of the positions in the vector. For every update, only one position in each of the vectors is updated. The estimate for two Fast-AGMS Sketches is computed by element-wise multiplying the vectors from both tables that use the same pair of random variables and hash function and summing them up. The final estimate is then given by the median of the per-vector estimates. Fast-AGMS Sketches only improve the update time of AGMS Sketches, the variance of the estimate is identical [43]. *Skimmed Sketches* [12] apply the idea of Bifocal Sampling (Section 2.2.1) to AGMS Sketches by treating dense and sparse join keys differently to improve the variance of the estimate. However, it remains to be shown that Fast-AGMS Sketches or Skimmed Sketches can be efficiently extended to n-way joins or selections.

Besides the AGMS-based estimators, the counting-based sketches *Fast-Count Sketches* [48] or *Count-Min Sketches* [8] may be used to compute estimates for two-way joins [43]. However, to the best of our knowledge no generalization exists that allows the computation of n-way joins or including selections.

Sketch-based estimation techniques may also be used for selectivity estimation in database systems by considering input relations as a stream. However, e.g. compared to simple samples (Section 2.2.1), constructing sketches requires us to know join attributes and selection attributes beforehand. Note that the AGMS approach does not only require memory for the sketch counters but also requires storing the seeds for the 4-wise independent hash functions, which are logarithmic in the number of distinct elements in the stream [42].

2.2.3 Graphical Models

Probabilistic graphical models are a popular technique that allows to express joint probability distributions in terms of graphs [3]. Bayesian Networks represent random variables as nodes and encode direct dependence between variables with a directed edge [3]. This way, every node is conditionally independent of its non-descendants given its parents. As any conditional or unconditional probability can be inferred from conditional probability tables for the nodes given their parents, constructing a Bayes Network can potentially reduce the amount of memory necessary to store the joint probability distribution.

Getoor et al. [15] have proposed computing the selectivity of joins subject to selections with equality predicates using Bayes Networks. Applying the idea of Bayesian Networks to selections on a single table is trivial, as every attribute can be seen as a random variable. Bayesian Networks for joins require using a trick: The Bayesian Network models the cross product of tables and an additional binary random variable - a *join indicator* - is introduced. These join indicators determine if a tuple from the cross product is included in a join. The algorithm is separated in an offline and an online phase. In the offline phase, the structure of the Bayesian Network is determined and the conditional probability tables are computed; in the online phase estimates are provided. While the approach shows promising estimation results, the construction of the model is fairly expensive: Finding the structure for a Bayesian Network is done by heuristic search through the space of possible Bayesian Networks by optimizing a score that requires executing grouped counts on base tables and join results. Sampling can be used to accelerate model construction, however, this comes with a loss of accuracy [15].

Tzoumas et. al. [50] built upon their idea but aimed for improved efficiency. Among their optimizations are restricting probability tables to two dimensions and using a model selection strategy inspired by *CORDS* algorithm [22] for detecting correlations and functional dependencies. Their approach does only require computing grouped counts for two-way joins and base-tables.

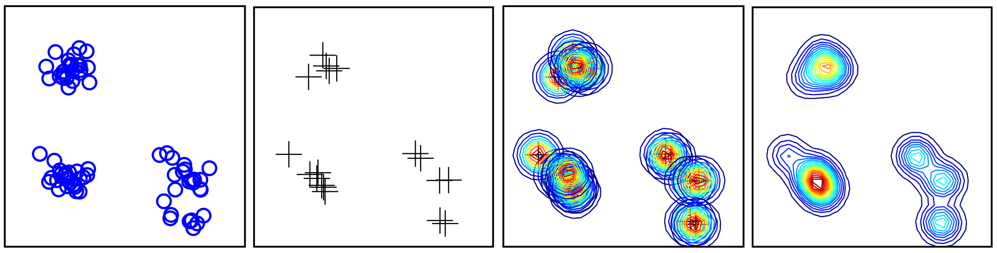
While probabilistic graphical models are a powerful technique for selectivity estimation in databases, their creation requires several passes over the input data, performing joins and computing grouped aggregates, which makes them more expensive to construct than the introduced sampling or sketch-based approaches. To the best of our knowledge there is no work on adapting probabilistic graphical models for selectivity estimation to changing data.

2.2.4 Wavelets

Wavelet transformations are a mathematical tool for hierarchically decomposing functions. They allow to describe functions by a coarse approximation function and detail coefficients influencing the functions at various scales [46]. Removing the coefficients with the lowest impact allows approximation with error guarantees. Chakrabarti et. al. [7] proposed using wavelets for approximate query processing - a problem that is related to selectivity estimation as it aims at providing better performance by delivering only approximate answers. Wavelet decomposition is applied to the joint frequency distribution of input relations, while the obtained *wavelet coefficients* are restricted to the most influential ones. The remaining coefficients form a *wavelet synopsis* that provides an approximation to the joint frequency distribution. This can be seen as applying lossy compression to the frequency distribution. The authors have shown that complex relational queries consisting of selections, projections, joins and aggregate queries can be computed without leaving the wavelet coefficient domain, which allows fast approximate answers to those queries. This way, the approximate result size of joins subject to filters can be obtained and used as a selectivity estimate. While wavelets have shown superior results compared to sampling, their construction is more expensive as it requires multiple passes over the data.

2.3 Kernel Density Estimators

Kernel Density Estimators are a well known tool from statistics for non-parametric estimation of probability density functions [44] using a random sample. Figure 2.1 visualizes how the estimate for a given two-dimensional distribution (Figure 2.1(a)) is obtained. Given a sample from the distribution (Figure 2.1(b)), local probability density functions - so called *kernels* - are centered on these sample points (Figure 2.1(c)). This can be seen as placing probability mass in the neighborhood of the sample points. The estimate is then given by averaging over the contributions of the local probability density functions (Figure 2.1(d)).



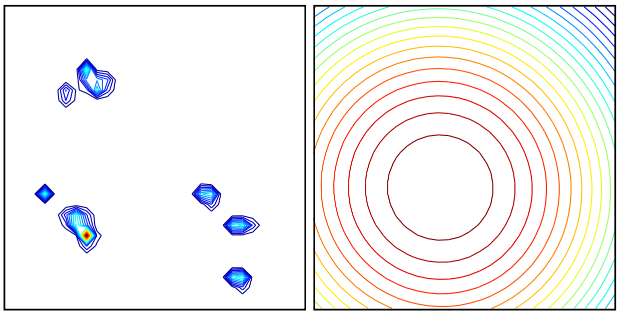
(a) Points in data set **(b)** Sampled points **(c)** Kernels **(d)** Estimated distribution

Figure 2.1: A Kernel Density Estimator approximates the underlying distribution of a given dataset (a) by picking a random sample (b), centering local probability distributions (kernels) around them (c), and averaging the local distributions (d). [21]

Formally, given a random sample $S = \{\vec{s}_1, ..., \vec{s}_n\} \subset \mathbb{R}$ drawn from d-variate vectors of continuous random variables with a probability density function $p : \mathbb{R}^d \to \mathbb{R}$, the kernel density estimate $\hat{p}_H(\vec{x}) : \mathbb{R}^d \to \mathbb{R}$ is given by:

$$\begin{aligned}\hat{p}_H(\vec{x}) &= \frac{1}{s}\sum_{i=1}^{s} K_H(\vec{t}^{(i)} - \vec{x}) \\ &= \frac{1}{s \cdot |H|}\sum_{i=1}^{s} K(H^{-1}[\vec{t}^{(i)} - \vec{x}])\end{aligned} \tag{2.13}$$

In this equation, $K : \mathbb{R}^d \to \mathbb{R}$ denotes the *kernel function*, which defines the shape of the local probability density functions. Any non-negative, symmetric, and real-valued function integrating to one is eligible as a kernel function. Note that, while the kernels follow a certain distribution, the resulting estimator does not assume any input distribution. In fact, the impact of the choice of the kernel function is negligible [44] and it can be selected based on desirable properties, e.g. continuous differentiability. Among popular choices are the *Gaussian* kernel,



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

Figure 2.2: Choosing the bandwidth has a crucial impact on estimation quality. If the bandwidth is too small (a), the estimator over-fits the sample. If it is too large (b), all local information is lost. [21]

which is a multivariate Gaussian distribution, and *Epanechnikov* kernel, which is a truncated second order polynomial. The parameter $H \in \mathbb{R}^{d \times d}$ is the *bandwidth matrix*, which is a smoothing parameter that controls the spread of the probability function. The matrix needs to be non-singular and symmetric. The bandwidth parameter has a major influence on the quality of the estimate. Figure 2.2 visualizes the effect of wrong choices for the bandwidth parameter: If the bandwidth parameter is selected too small (Figure 2.2(a)), the estimated distribution over-fits the sample; If the sample bandwidth parameter is chosen too large (Figure 2.2(b)), the estimator under-fits the sample and the local information of the sample points is lost. It is a common simplification to restrict H to diagonal matrices with $H_{i,i} = h_i > 0$.

Finding a suitable value for the bandwidth matrix is a non-trivial task that is performed by a *bandwidth selector*. The problem has been under heavy investigation by the statistics community [27, 44]. Most methods aim at minimizing the *mean integrated squared error (MISE)* which is $\mathbb{E}\left[\int_{\vec{x}} \left(p\left(\vec{x}\right) \hat{p}_H\left(\vec{x}\right)\right)^2 d\vec{x}\right]$. The problem is particularly challenging, as the true output cardinality $p(\vec{x})$ is usually unknown. Therefore, bandwidth selectors optimize H over an approximation of the true density. *Scotts rule* [44] assumes the true distribution to be a multivariate normal distribution $\mathcal{N}\left(\mu, \Sigma\right)$ and delivers a closed-form formula for an optimal diagonal bandwidth matrix H that depends on the per-dimension standard deviation and dimensionality of the sample.

$$\hat{h}_i^{scott} = s^{-\frac{1}{d+4}} \cdot \sigma_i \tag{2.14}$$

While easy to calculate, this formula potentially introduces large estimation errors if the assumption is violated. More sophisticated classes of bandwidth selectors obtain there approximation through *Cross Validation* on the sample or *Plug-In* methods iteratively refining a pilot distribution. These approximations deliver significantly better results for general distributions, but are more computationally expensive [27, 17].

2.3.1 Kernel Density Estimators for Range Selectivity Estimation

Multiple authors have proposed Kernel Density Estimators for selectivity estimation of multi-dimensional range queries over real-valued attributes [4, 17]. The key idea is to draw the sample from the relation R and integrate the Kernel Density Estimator over the query range. Based on a KDE constructed from a uniform sample S from R, we can calculate the estimated selectivity of a (hyper-rectangular) query region Ω by integrating $\hat{p}_H(\vec{x})$:

$$\hat{p}_H(\Omega) = \frac{1}{s} \sum_{i=1}^{s} \underbrace{\int_{\Omega} \frac{K\left(H^{-1}\left[\vec{t}^{(i)} - \vec{x}\right]\right)}{|H|} d\vec{x}}_{=\hat{p}_H^{(i)}(\Omega)} \tag{2.15}$$

Using a Kernel Density Estimator for selectivity estimation has several benefits:

- Kernel Density Estimators do not require the independence assumption.
- Kernel Density Estimators have been shown to converge faster to the underlying distribution than histograms do [44] and to provide better results [17] compared to plain sampling-based estimators [29, 34].
- Kernel Density Estimators models are easily constructed as they only require a simple sample. Compared to state of the art multi-dimensional histograms ([5, 17]), they do not require special data structures for their representation nor the additional computations for their construction.
- Kernel Density Estimators are adaptive with respect to changes to the input relation, as maintaining a sample of a relation that is subject to insertions, deletions and updates is a well-researched topic [16].

In a previous publication [21], we have shown that the bandwidth selection problem can be solved more efficiently in a selectivity estimation setup. As a database system usually executes a query after the optimizer requested selectivity estimates, the true output cardinality of operators in the execution plan are available. This information can be used to select a bandwidth that optimizes the quality for previously observed queries instead of optimizing the *MISE*. Formally, given a loss function $\mathcal{L} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and a set of representative queries $Q = \{\Omega_1, \ldots, \Omega_q\}$, the bandwidth selector for a diagonal bandwidth matrix solves the following optimization problem:

$$\begin{aligned} H^* = & \operatorname*{argmin}_{H} \frac{1}{q} \sum_{i=1}^{q} \mathcal{L}\left(\hat{p}_H(\Omega_i), \frac{\left|\sigma_{\vec{x}\in\Omega_i}(R)\right|}{|R|}\right) \\ & \text{s.t. } \forall i : h_i > 0 \end{aligned} \tag{2.16}$$

The loss function can be any loss function like the quadratic, absolute or relative error. If the objective function is continuously differentiable with respect to the bandwidth components h_i, the optimization problem can be solved by a gradient-based bound-constrained optimization algorithm. To do so, we need to derive the gradient $\nabla_H \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial h_1}, \ldots, \frac{\partial \mathcal{L}}{\partial h_d}\right]^T$. Using the chain-rule yields the following equation for the partial derivatives:

$$\frac{\partial \mathcal{L}}{\partial h_i} = \frac{\partial \mathcal{L}}{\partial \hat{p}_H(\Omega)} \cdot \frac{\partial \hat{p}_H(\Omega)}{\partial h_i} \tag{2.17}$$

We therefore need to select the kernel function continuously-differentiable with respect to the bandwidth on the optimization region as well as the loss function continuously differentiable with respect to the estimate $\hat{p}_{H(\Omega)}$. The *Gaussian Kernel*, which is a multivariate normal distribution with zero mean, and the quadratic error $\mathcal{L}_{\text{Quadratic}} = \left(\hat{p}_H(\Omega_i) - \frac{\left|\sigma_{\vec{x}\in\Omega_i}(R)\right|}{|R|}\right)^2$ are valid choices we have evaluated in our previous publication. We suggested two optimization strategies:

Batch: Collect a fixed-size sample from the relation R, set the initial bandwidth using Scotts rule, collect a set of queries in a ring buffer and optimize the bandwidth using an offline optimization algorithm. Increasing the number of queries leads to a more robust optimization, while decreasing reduces the number of computations necessary to calculate estimates and partial derivatives.

Adaptive: Collect a fixed-size sample from the relation R, set the initial bandwidth using Scotts rule and use an online optimization algorithm to apply changes to the bandwidth in mini-batches of few queries.

Furthermore, we suggested to accelerate estimation and bandwidth optimization by using a GPU. This allows us to evaluate much larger samples in the same time budget compared to a CPU implementation. To do so, the sample and initial bandwidth are transferred to GPU device memory once at creation of the estimator.

If an estimate is requested, the query bounds are transferred to the device memory. The local contribution $\hat{p}_H^{(i)}(\Omega)$ for every sample point in Equation 2.15 can be computed independently by different threads and stored in a buffer. Applying a parallel binary reduction scheme allows to efficiently aggregate the individual contributions to the final estimate. The estimate is then transferred back to the host for usage in query optimization.

After query execution finished, the observed actual selectivity is transferred to the device memory. The computation of the partial derivatives follows a structure similar to the estimation computations. Further steps depend on the optimization strategy. If the batch strategy is used and sufficient queries were collected, we use an off-the-shelf implementation of *LBFGS-B*

[6] to perform the numerical optimization. Until converged, the optimization algorithm proposes a bandwidth in every iteration that is transferred to the device. The GPU computes objective and gradient, which are afterwards transferred back to the host memory.

If the adaptive optimization strategy is used and a mini-batch of queries was collected, we use a GPU-implementation of the *RMSprop* [49] algorithm to adjust the bandwidth. Compared to the batch strategy, this has several benefits: The computation of the partial derivative of the kernel with respect to the bandwidth does not depend on the actual selectivity. Thus, we can perform its computation while the query is still running and only need to perform a scalar multiplication with the derivative of the loss function afterwards. Bandwidth and gradient do not have to be transferred to the host as the optimization algorithm operates entirely on the GPU. However, these benefits come with a less robust optimization.

Additionally, we introduced a GPU-friendly error-driven sample maintenance algorithm [21, 20] that assesses the usefulness of sample points by computing the leave-one-out estimate for each of them. These estimates are translated to a score that assesses the contribution of every sample point. Sample points that constantly improve the estimates by their absence are likely misrepresenting the distribution in the relation and are replaced by randomly drawn tuples. Using this approach, we can maintain a reasonable sample quality without having to mirror changes to the relation set to the sample on the GPU.

We made the following findings:

- Kernel Density Estimators with error-driven bandwidth selection outperform naive bandwidth selection using Scotts rule with respect to estimation quality
- Kernel Density Estimators with error-driven bandwidth selection and sample maintenance outperform state-of-the-art multidimensional histograms with respect to estimation quality for static and changing relations
- Error-driven bandwidth selection provides estimates comparable to state-of-the-art cross-validation techniques.
- The batch strategy provides more robust estimates compared to the adaptive approach.
- GPU-accelerated Kernel Density Estimators scale up to very large sample sizes without a significant performance impact.

These positive results motivate us to investigate, if we can apply Kernel Density Estimators to even more database operations. In this thesis, we will investigate Kernel Density Estimators for discrete data types. In particular, we estimate the selectivities for selections with conjunctive equality predicates and equi-joins subject to such selections.

Part III

Approach

3 Approach

In this Chapter, we introduce our approach to join selectivity estimation using Kernel Density Estimators. Our basic idea is to model the joint frequency distribution in base tables using Kernel Density Estimators and then exploiting these models to estimate the selectivity of joins.

As the joint frequency distribution is discrete and our previous approach constructs a continuous probability density function, we introduce the *categorical kernel* in Section 3.1 that provides a discrete probability mass function instead. We show how these models can be used to provide selectivity estimates for selections consisting of conjunctive equality predicates and provide the gradient required for error-driven bandwidth optimization.

In Section 3.2, we explain how the selectivity of joins can be computed using joint frequency distributions. In Section 3.3, we show how and under which assumptions the estimated joint frequency distribution provided by continuous Kernel Density Estimators can be used to estimate the selectivity of two-way joins efficiently. Furthermore, we provide the gradient to perform error-driven bandwidth optimization. We give a sequential algorithm for the computation of join selectivity estimate and their error-gradient in Section 3.4. Moreover, we sketch a possible parallel implementation on a GPU.

Section 3.5 presents possibilities to efficiently extend the estimator to n-way joins. Finally, in Section 3.6, we discuss the relationship of our estimator to simple samples and the independence and uniformity assumption.

3.1 The Categorical Kernel for Discrete Attributes

The kernels we used in our previous work deliver estimates for a continuous probability density [21] . This makes them suitable for estimating range-queries over continuous attributes. However, they are not directly applicable to discrete attributes such as integer keys. Li and Racine [31] proposed a kernel for distributions over categorical random variables. Given a finite attribute domain C and the bandwidth parameter $\lambda \in [0, 1)$, the categorical kernel k_λ :

$C \times C \rightarrow [0, 1]$ is defined as:

$$k_\lambda (x_1, x_2) = \begin{cases} (1-\lambda) & x_1 = x_2 \\ \frac{\lambda}{|C|-1} & x_1 \neq x_2 \end{cases} \tag{3.1}$$

The kernel contributes $(1 - \lambda)$ if sample point and function value coincide, the remaining probability mass is evenly distributed across all remaining values in C. Figure 3.1 shows the categorical kernel on the value 2 with $\lambda = 0.7$ and $C = \{0, 1, 2, 3, 4\}$.

The kernel is a *product kernel*, which means that for a joint probability density over multiple attributes $C = C_1 \times ... \times C_d$ the contribution of a sample point $\vec{s} \in C$ for a vector of smoothing parameters $\vec{\lambda}$ is given by the product of the single-dimensional kernels.

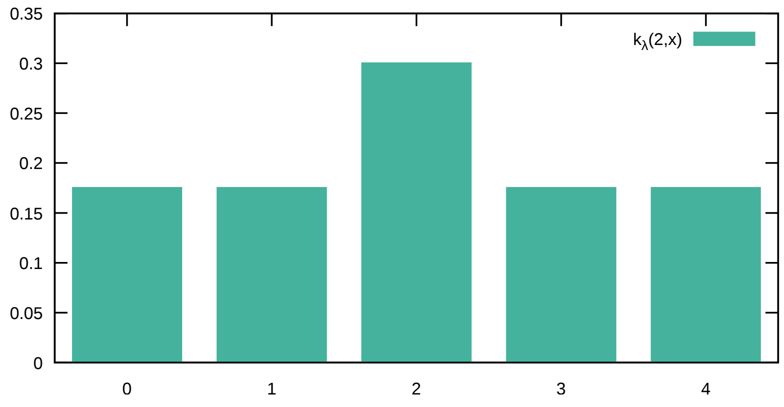
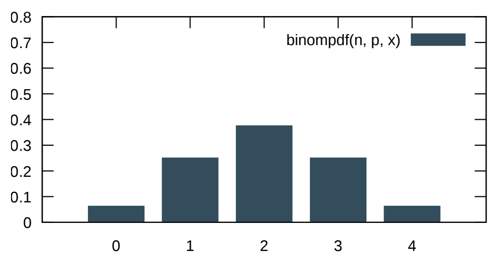


Figure 3.1: The categorical kernel on value 2 with $\lambda = 0.7$ and $C = \{0, 1, 2, 3, 4\}$

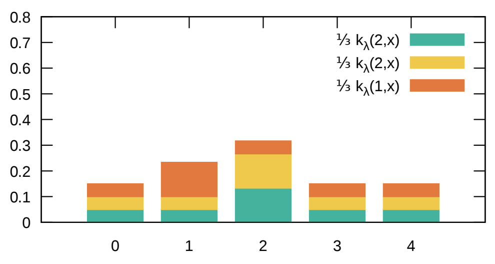
$$K_{\vec{\lambda}} (\vec{s}, \vec{x}) = \prod_{i=1}^{d} k_{\lambda_i} (s_i, x_i) \tag{3.2}$$

Like in the continuous case, the value of the bandwidth parameter λ is essential for the quality of the estimate. If the smoothing parameter approaches zero for a dimension, the estimated probability along this dimension approaches the relative frequency distribution in the sample. A smoothing parameter approaching one removes probability mass from the sample points and places more probability mass on values not included in the sample. Figure 3.2 shows a one-dimensional example of a discrete Kernel Density Estimator fitting the sample $S = \{2, 2, 3\}$ drawn from a binomial distribution with $n = 4$ and $p = 0.5$: (a) Shows the target distribution; The remaining pictures show the estimated probability density and visualize the contribution of every sample point for fitting (b), over-fitting (c), and under-fitting value of λ (d).

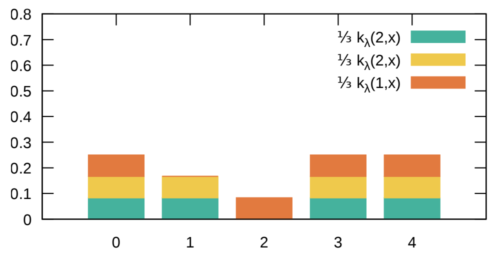
The estimated distribution may even be a mix of discrete $C = C_1 \times ... \times C_d$ and continuous



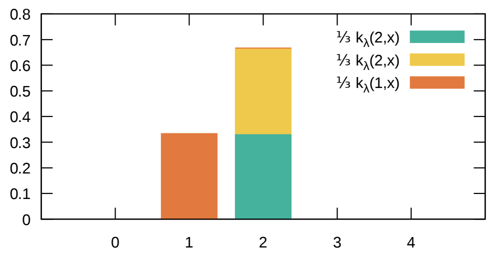
(a) Binomial distribution



(b) Fitted estimate ($\lambda = 0.4$)



(c) Under-fitted estimate ($\lambda \approx 1.0$)



(d) Over-fitted estimate ($\lambda \approx 0.0$)

Figure 3.2: Effect of the bandwidth parameter λ on a Kernel Density Estimator obtained by sampling from a binomial distribution with $n = 4$ and $p = 0.5$.

attributes $\Omega \subseteq \mathbb{R}^r$. Given a sample $S = \left\{ \left(\vec{s}^{(1)}, \vec{t}^{(1)} \right), ..., \left(\vec{s}^{(n)}, \vec{t}^{(n)} \right) \right\} \subseteq C \times \Omega$ and an arbitrary continuous kernel $K_H : \mathbb{R}^r \to \mathbb{R}$ with a bandwidth matrix H, the estimate $\hat{p} : C \times \Omega \to \mathbb{R}$ is computed by multiplying the discrete and continuous contributions of every sample point.

$$\hat{p}\left(\vec{c}, \vec{\omega}\right) = \frac{1}{n} \sum_{i=1}^{n} K_{\vec{\lambda}} \left(\vec{s}^{(i)}, \vec{c} \right) \cdot K_H \left(\vec{t}^{(i)}, \vec{\omega} \right) \tag{3.3}$$

The kernel k_λ and the obtained estimate is continuously differentiable with respect to λ, which is required to perform error-driven optimization of bandwidth parameter. The gradient of the kernel k_λ is given by:

$$\frac{\partial k\left(x_1, x_2\right)}{\partial \lambda_l} = \begin{cases} -1 & x_1 = x_2 \\ \frac{1}{|C|-1} & x_1 \neq x_2 \end{cases} \tag{3.4}$$

The gradient of the estimate $\hat{p}\left(\vec{c}, \vec{\omega}\right)$ with respect to a component λ_l of the discrete bandwidth vector $\vec{\lambda}$ is given by:

$$\begin{aligned}\frac{\partial \hat{p}\left(\vec{c},\vec{\omega}\right)}{\partial \lambda_l} &= \frac{\partial}{\partial \lambda_l}\frac{1}{n}\sum_{i=1}^{n} K_{\vec{\lambda}}\left(\vec{s}^{(i)},\vec{c}\right)\cdot K_H\left(\vec{t}^{(i)},\vec{\omega}\right)\\ &= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\partial}{\partial \lambda_l}K_{\vec{\lambda}}\left(\vec{s}^{(i)},\vec{c}\right)\right)\cdot K_H\left(\vec{t}^{(i)},\vec{\omega}\right)\\ &= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\partial}{\partial \lambda_l}k_{\lambda_l}\left(s_l^{(i)},x_l\right)\right)\cdot\left(\prod_{j\neq l}k_{\lambda_j}\left(s_j^{(i)},x_j\right)\right)\cdot K_H\left(\vec{t}^{(i)},\vec{\omega}\right)\end{aligned} \tag{3.5}$$

Note that the continuous kernel only participates as a factor to the gradient of the discrete bandwidth. The discrete contributions participate as a factor in the gradient with respect to the bandwidth matrix H.

$$\begin{aligned}\frac{\partial \hat{p}\left(\vec{c},\vec{\omega}\right)}{\partial H} &= \frac{\partial}{\partial H}\frac{1}{n}\sum_{i=1}^{n} K_{\vec{\lambda}}\left(\vec{s}^{(i)},\vec{c}\right)\cdot K_H\left(\vec{t}^{(i)},\vec{\omega}\right)\\ &= \frac{1}{n}\sum_{i=1}^{n} K_{\vec{\lambda}}\left(\vec{s}^{(i)},\vec{c}\right)\cdot\left(\frac{\partial}{\partial H}K_H\left(\vec{t}^{(i)},\vec{\omega}\right)\right)\end{aligned} \tag{3.6}$$

In the following, we will not cover mixtures of discrete and continuous models further and focus on discrete models. Note that this is merely due to notational simplicity and space constraints - the findings of the following Sections are easily extended to mixed Kernel Density Estimators that allow additional continuous range queries on attributes.

3.1.1 Using the Categorical Kernel for Selections with Equality Predicates

Using the categorical kernel for selections with equality predicates on the attributes of a relational table is straightforward as a discrete probability distribution function already models the probability of a vector from the domain. Given a relation $R = (A_1, ..., A_d)$ with attributes that are considered discrete, a sample $S = \left\{\vec{s}^{(1)}, ..., \vec{s}^{(n)}\right\} \subseteq R$ and a bandwidth vector $\vec{\lambda} \in [0,1)^d$, the probability for selection σ of conjunctive equality predicates is computed by multiplying the contributions for attributes participating in the selection.

$$\hat{p}_{\vec{\lambda}}\left(\sigma\right) = \frac{1}{n}\sum_{i=1}^{n}\prod_{(A_j=x_j)\in\sigma} k_{\lambda_j}\left(s_j^{(i)},x_j\right) \tag{3.7}$$

Note that evaluating the categorical kernel requires the number of distinct values in the attribute, which is a common statistic in query optimizers. The kernel is only defined on its domain, hence, a selectivity of zero should be returned whenever an equality predicate targets a value that is not part of the attribute domain. However, making this decision in general requires scanning the relation or an index, which is unfeasible for selectivity estimation. Therefore, we have to assume that all incoming values are a part of the domain. In case additional statistics are maintained, i.e. minimum and maximum values or a bloom filter, these can be used to reduce the error introduced by this assumption.

The gradient for a given selection with respect to λ_l only has a non-zero value when A_l participates in the selection.

$$\frac{\partial \hat{p}_{\vec{\lambda}}(\sigma)}{\partial \lambda_l} = \begin{cases} \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\partial}{\partial \lambda_l} k_{\lambda_l}\left(s_l^{(i)}, x_l\right)\right)\prod_{\left(A_j = x_j\right)\in\sigma \wedge l \neq j} k_{\lambda_j}\left(s_j^{(i)}, x_j\right) & \left(A_l = x_l\right) \in \sigma \\ 0 & \left(A_l = x_l\right) \notin \sigma \end{cases} \tag{3.8}$$

3.2 Probabilistic View on Joins

The output cardinality of equi-joins and selections can be computed based on the frequency distributions in the base tables. We denote the join selectivity for a given join query q by $p(q)$, which is the output cardinality relative to the Cartesian product. Further, we refer to the relative frequency distribution in a relation R as p_R. Given two relations R_1 and R_2 being joined on their first attribute without loss of generality, we can calculate the output cardinality of the join by iterating over the intersection of the join attribute domains.

$$\begin{aligned} q_1 &= R_1 \bowtie_{R_1.A_1 = R_2.A_1} R_2 \\ p(q_1) &= \sum_{v \in R_1.A_1 \cap R_2.A_1} p_{R_1}(A_1 = v)\, p_{R_2}(A_1 = v) \end{aligned} \tag{3.9}$$

Using the set intersection of $R_1.A_1$ and $R_2.A_1$ is necessary because the domains may not be equal and we do not assume the frequency distributions to be defined outside the attribute domains.

When additional selections σ_1 and σ_2 are applied to the attributes of R_1 and R_2, we interpret them as sets of conjunctive predicates that are passed to the relative frequency distributions.

$$\begin{aligned} q_2 &= \sigma_1(R_1) \bowtie_{R_1.A_1 = R_2.A_1} \sigma_2(R_2) \\ p(q_2) &= \sum_{v \in R_1.A_1 \cap R_2.A_1} p_{R_1}(A_1 = v \wedge \sigma_1)\, p_{R_2}(A_1 = v \wedge \sigma_2) \end{aligned} \tag{3.10}$$

In case there is a selection with an equality predicate on one of the join keys, assume it is $R_1.A_1 = v$, there is no need to sum over all values in the intersection.

$$p(q_2) = p_{R_1}(A_1 = v \wedge \sigma_1)\, p_{R_2}(A_1 = v \wedge \sigma_2) \tag{3.11}$$

If the query is extended by an additional selection and join on table R_3 either of the following two cases apply: (1) If one of the join attributes was already part of the previous join, the attribute domain participates in the intersection. Without loss of generality, say $R_2.A_1$ joins with $R_3.A_1$.

$$\begin{aligned} q_3 &= \sigma_1(R_1) \bowtie_{R_1.A_1=R_2.A_1} \sigma_2(R_2) \bowtie_{R_2.A_1=R_3.A_1} \sigma_3(R_3) \\ p(q_3) &= \sum_{v \in R_1.A_1 \cap R_2.A_1 \cap R_3.A_1} p_{R_1}(A_1 = v \wedge \sigma_1)\, p_{R_2}(A_1 = v \wedge \sigma_2)\, p_{R_3}(A_1 = v \wedge \sigma_3) \end{aligned} \tag{3.12}$$

(2) If the join attribute matches a different attribute in the previous relations, the Cartesian product of the attribute intersections has to be considered. Without loss of generality, say $R_2.A_2$ joins with $R_3.A_1$.

$$\begin{aligned} q_4 &= \sigma_1(R_1) \bowtie_{R_1.A_1=R_2.A_1} \sigma_2(R_2) \bowtie_{R_2.A_2=R_3.A_1} \sigma_3(R_3) \\ p(q_4) &= \sum_{\substack{v_1 \in R_1.A_1 \cap R_2.A_1; \\ v_2 \in R_2.A_2 \cap R_3.A_1}} p_{R_1}(A_1 = v_1 \wedge \sigma_1)\, p_{R_2}(A_1 = v_1 \wedge A_2 = v_2 \wedge \sigma_2)\, p_{R_3}(A_1 = v_2 \wedge \sigma_3) \end{aligned} \tag{3.13}$$

Repeatedly applying this scheme allows us to calculate the selectivity of arbitrary equi-joins queries with selections.

3.3 Using the Categorical Kernel for Two-Way Joins

The categorical kernel enables us to calculate the selectivity of two-way joins subject to optional selections with conjunctive equality predicates on base tables. Naively, we can calculate the join selectivity by substituting the exact distribution functions p_{R_1} and p_{R_2} in Equation 3.10 by the estimated distributions $\hat{p}_{R_1}$ and $\hat{p}_{R_2}$ provided by Kernel Density Estimators as described in Section 3.1. The discrete Kernel Density Estimator $\hat{p}_{R_1}$ is specified by the sample $S_1 = \{\vec{s}^{(1)}, ..., \vec{s}^{(m)}\}$ and the bandwidth vector $\vec{\lambda}$; The estimator $\hat{p}_{R_2}$ is specified by the sample $S_2 = \{\vec{t}^{(1)}, ..., \vec{t}^{(n)}\}$ and the bandwidth vector $\vec{\gamma}$.

$$\begin{aligned} q &: \sigma_1(R_1) \bowtie_{R_1.A_1=R_2.A_1} \sigma_2(R_2) \\ \hat{p}(q) &= \sum_{v \in R_1.A_1 \cap R_2.A_1} \hat{p}_{R_1}(A_1 = v \wedge \sigma_1)\, \hat{p}_{R_2}(A_1 = v \wedge \sigma_2) \end{aligned} \tag{3.14}$$

The join attributes are assumed to not take part in the selections, as Equation 3.11 shows this case is trivial and reduces to a multiplication of two base table estimates. Evaluating the equation in this form is not suitable for selectivity estimation: Selecting an algorithm for computing the set intersection between the join attributes raises the same optimization problem as selecting a join algorithm. Moreover, computing this intersection would be as expensive as computing the join itself. Similar to the case of point queries in Section 3.1.1, we introduce an assumption relieving us from computing the intersection:

Assumption: The domains of the join attributes are equal, i.e. $R_1.A_1 = R_2.A_1 = A$.

We will discuss the severity of this assumption in Section 3.7. Furthermore, properties of the categorical kernel have to be utilized to calculate estimates efficiently. First, we substitute the distributions by their definition equivalent to Equation 3.1.1.

$$\begin{aligned}\hat{p}(q) &= \sum_{v \in A} \left(\left(\frac{1}{n} \sum_{i=1}^{m} k_{\lambda_j}\left(s_j^{(i)}, v\right) \underbrace{\prod_{(A_j = x_j) \in \sigma_1} k_{\lambda_j}\left(s_j^{(i)}, x_j\right)}_{r_1^{(i)}} \right) \left(\frac{1}{m} \sum_{i=1}^{n} k_{\gamma_j}\left(t_j^{(i)}, v\right) \underbrace{\prod_{(A_j = x_j) \in \sigma_2} k_{\gamma_j}\left(t_j^{(i)}, x_j\right)}_{r_2^{(i)}} \right) \right) \\ &= \sum_{v \in A} \left(\frac{1}{m \cdot n} \left(\sum_{i=1}^{m} k_{\lambda_j}\left(s_j^{(i)}, v\right) r_1^{(i)} \right) \left(\sum_{i=1}^{n} k_{\gamma_j}\left(t_j^{(i)}, v\right) r_2^{(i)} \right) \right)\end{aligned} \tag{3.15}$$

Note that the variable indexed by the outer sum only affects the kernels on the join attribute. The invariant contribution of the kernels on the selection predicates is factored out for every sample point $s^{(j)}$ from S_k and summarized by $r_k^{(j)}$. Note that in case of no selection, i.e. $\sigma_k = \{\}$, $r_k^{(j)}$ is always one.

Finally, we remove the need to sum over the entire set of distinct values in A by using the property that the categorical kernel assigns the same probability if its arguments are not equal. From this property follows that the Kernel Density Estimators in Equation 3.15 will assign the same probability for every v that is not included in either of the samples. We denote the unnormalized estimates for join values that are not in S_k by the quantity Ω_k.

$$\begin{aligned}\hat{p}(q) = &\sum_{v \in S_1.A_1 \cup S_2.A_1} \left(\frac{1}{m \cdot n} \left(\sum_{i=1}^{m} k_{\lambda_j}\left(s_j^{(i)}, v\right) r_1^{(i)} \right) \left(\sum_{i=1}^{n} k_{\gamma_j}\left(t_j^{(i)}, v\right) r_2^{(i)} \right) \right) \\ &+ \frac{\left|A \setminus (S_1.A_1 \cup S_2.A_1)\right|}{m \cdot n} \underbrace{\left(\frac{\lambda_1}{|A| - 1} \sum_{i=1}^{m} r_1^{(i)} \right)}_{\Omega_1} \underbrace{\left(\frac{\gamma_1}{|A| - 1} \sum_{i=1}^{n} r_2^{(i)} \right)}_{\Omega_2}\end{aligned} \tag{3.16}$$

Observing the relationship between Ω_1 and the value of the first inner summer for a value v, we find that they only differ in the contributions for sample points i with $s_1^{(i)} = v$. The same relationship holds for Ω_2 and second inner sum. This allows us to substitute the sums over all sample points by equivalent representations using Ω_1 and Ω_2 and sums over disjoint subsets of the samples.

$$\begin{aligned}\hat{p}\left(q\right) = &\sum_{v\in(S_1.A_1\cup S_2.A_1)}\left(\frac{1}{m\cdot n}\left(\Omega_1+(1-\lambda_1)\sum_{i:s_1^{(i)}=v}r_1^{(i)}-\frac{\lambda_1}{|A|-1}\sum_{i:s_1^{(i)}=v}r_1^{(i)}\right)\right.\\ &\left.\left(\Omega_2+(1-\gamma_1)\sum_{i:t_1^{(i)}=v}r_1^{(i)}-\frac{\gamma_1}{|A|-1}\sum_{i:t_1^{(i)}=v}r_2^{(i)}\right)\right)+\frac{\left|A\backslash(S_1.A_1\cup S_2.A_1)\right|}{m\cdot n}\cdot\Omega_1\cdot\Omega_2\\ = &\sum_{v\in(S_1.A_1\cup S_2.A_1)}\left(\frac{1}{m\cdot n}\left(\Omega_1+\left(1-\lambda_1-\frac{\lambda_1}{|A|-1}\right)\sum_{i:s_1^{(i)}=v}r_1^{(i)}\right)\right.\\ &\left.\left(\Omega_2+\left(1-\gamma_1-\frac{\gamma_1}{|A|-1}\right)\sum_{i:t_1^{(i)}=v}r_2^{(i)}\right)\right)+\frac{\left|A\backslash(S_1.A_1\cup S_2.A_1)\right|}{m\cdot n}\cdot\Omega_1\cdot\Omega_2\end{aligned}\tag{3.17}$$

This is the underlying equation for the estimation algorithm introduced in Section 3.4.1.

As we want to perform error-driven optimization of the bandwidth parameter with a gradient-based optimization algorithm, we need to compute the gradient for the previous Equation with respect to the bandwidth operators as well. The gradient for the bandwidth on the join attributes is given by:

$$\begin{aligned}\frac{\partial}{\partial\lambda_1}\hat{p}\left(q\right) = &\sum_{v\in(S_1.A_1\cup S_2.A_1)}\left(\frac{1}{m\cdot n}\left(\frac{\partial}{\partial\lambda_1}\Omega_1+\left(-1-\frac{1}{|A|-1}\right)\sum_{i:s_1^{(i)}=v}r_1^{(i)}\right)\right.\\ &\left.\left(\Omega_2+\left(1-\gamma_1-\frac{\gamma_1}{|A|-1}\right)\sum_{i:t_1^{(i)}=v}r_2^{(i)}\right)\right)+\frac{\left|A\backslash(S_1.A_1\cup S_2.A_1)\right|}{m\cdot n}\cdot\left(\frac{\partial}{\partial\lambda_1}\Omega_1\right)\cdot\Omega_2\end{aligned}\tag{3.18}$$

$$\begin{aligned}\frac{\partial}{\partial\gamma_1}\hat{p}\left(q\right) = &\sum_{v\in(S_1.A_1\cup S_2.A_1)}\left(\frac{1}{m\cdot n}\left(\Omega_1+\left(1-\lambda_1-\frac{\lambda_1}{|A|-1}\right)\sum_{i:s_1^{(i)}=v}r_1^{(i)}\right)\right.\\ &\left.\left(\frac{\partial}{\partial\gamma_2}\Omega_2+\left(-1-\frac{1}{|A|-1}\right)\sum_{i:t_1^{(i)}=v}r_2^{(i)}\right)\right)+\frac{\left|A\backslash(S_1.A_1\cup S_2.A_1)\right|}{m\cdot n}\cdot\Omega_1\cdot\left(\frac{\partial}{\partial\gamma_2}\Omega_2\right)\end{aligned}\tag{3.19}$$

The gradient for Ω_1 and Ω_2 is given by the following Equations. Restricting the bandwidth parameters to $\lambda_l,\gamma_l\in(0,1)$ allows us to compute the gradient directly from previously computed Ω_1 and Ω_2.

$$\begin{aligned}\frac{\partial}{\partial\lambda_1}\Omega_1 &= \frac{1}{|A|-1}\sum_{i=1}^{m}r_1^{(i)}\\ &\underset{\lambda_1\neq 0}{=}\frac{\Omega_1}{\lambda_1}\end{aligned}\tag{3.20}$$

$$\begin{aligned}\frac{\partial}{\partial\gamma_1}\Omega_2 &= \frac{1}{|A|-1}\sum_{i=1}^{n} r_2^{(i)} \\ &\underset{\gamma_1\neq 0}{=} \frac{\Omega_2}{\gamma_1}\end{aligned} \tag{3.21}$$

The gradient with respect to the other bandwidth components for attributes participating in the selections are given by the following equations:

Case $(\lambda_l = x_l) \in \sigma_1$:

$$\begin{aligned}
\frac{\partial}{\partial\lambda_l}\hat{p}\left(q\right) &= \sum_{v\in(S_1.A_1\cup S_2.A_1)}\left(\frac{1}{m\cdot n}\left(\frac{\partial}{\partial\lambda_l}\Omega_1 + \left(1-\lambda_1-\frac{\lambda_1}{|A|-1}\right)\sum_{i:s_1^{(i)}=v}\frac{\partial}{\partial\lambda_l}r_1^{(i)}\right)\right. \\
&\quad \left.\left(\Omega_2 + \left(1-\gamma_1-\frac{\gamma_1}{|A|-1}\right)\sum_{i:t_1^{(i)}=v} r_2^{(i)}\right)\right) \\
&\quad + \frac{\left|A\backslash(S_1.A_1\cup S_2.A_1)\right|}{m\cdot n}\cdot\left(\frac{\partial}{\partial\lambda_l}\Omega_1\right)\cdot\Omega_2 \\
\frac{\partial}{\partial\lambda_l}\Omega_1 &= \frac{\lambda_1}{|A|-1}\sum_{i=1}^{n}\frac{\partial}{\partial\lambda_l}r_1^{(i)}\left(\sigma_1\right) \\
\frac{\partial}{\partial\lambda_l}r_1^{(i)} &= \left(\frac{\partial}{\partial\lambda_l}k_{\lambda_l}\left(s_j^{(i)},x_l\right)\right)\prod_{(A_j=x_j)\in\sigma_1:j\neq l}k_{\lambda_j}\left(s_j^{(i)},x_j\right) \\
&\underset{\lambda_l\neq 0}{=} \begin{cases}-\frac{r_1^{(i)}}{1-\lambda_l} & s_j^{(i)} = x_l \\ \frac{r_1^{(i)}}{\lambda_l} & s_j^{(i)} \neq x_l\end{cases}
\end{aligned}$$

Case $(\lambda_l = x_l) \notin \sigma_1$:

$$\frac{\partial}{\partial\lambda_l}\hat{p}\left(q\right) = 0 \tag{3.22}$$

Case $(\gamma_l = x_l) \in \sigma_2$:

$$\begin{aligned}
\frac{\partial}{\partial \gamma_l}\hat{p}\left(q\right) &= \sum_{v \in (S_1.A_1 \cup S_2.A_1)} \left(\frac{1}{m \cdot n} \left(\Omega_1 + \left(1 - \lambda_1 - \frac{\lambda_1}{|A|-1}\right) \sum_{i: s_1^{(i)} = v} r_1^{(i)} \right) \right. \\
&\quad \left. \left(\frac{\partial}{\partial \gamma_l}\Omega_2 + \left(1 - \gamma_1 - \frac{\gamma_1}{|A|-1}\right) \sum_{i: t_1^{(i)} = v} \frac{\partial}{\partial \gamma_l} r_2^{(i)} \right) \right) \\
&\quad + \frac{\left|A \setminus (S_1.A_1 \cup S_2.A_1)\right|}{m \cdot n} \cdot \Omega_1 \cdot \left(\frac{\partial}{\partial \gamma_l}\Omega_2 \right) \\
\frac{\partial}{\partial \gamma_l}\Omega_2 &= \frac{\gamma_1}{|A|-1} \sum_{i=1}^{n} \frac{\partial}{\partial \gamma_l} r_1^{(i)} \qquad (3.23) \\
\frac{\partial}{\partial \gamma_l} r_2^{(i)} &= \left(\frac{\partial}{\partial \gamma_l} k_{\lambda_l}\left(t_j^{(i)}, x_l\right) \right) \prod_{(A_j = x_j) \in \sigma_2 : j \neq l} k_{\lambda_j}\left(t_j^{(i)}, x_j\right) \\
&\underset{\gamma_l \neq 0}{=} \begin{cases} -\frac{r_2^{(i)}}{1-\gamma_l} & t_j^{(i)} = x_l \\ \frac{r_2^{(i)}}{\gamma_l} & t_j^{(i)} \neq x_l \end{cases}
\end{aligned}$$

Case $(\gamma_l = x_l) \notin \sigma_2$:

$$\frac{\partial}{\partial \gamma_l}\hat{p}\left(q\right) = 0$$

The gradients $\frac{\partial}{\partial \lambda_i} r_1^{(i)}$ and $\frac{\partial}{\partial \gamma_i} r_2^{(i)}$ are computed by substituting the kernel on attributes $S_1.A_j$ and $S_2.A_j$ respectively with the gradient given in Equation 3.4. Note that these derivatives can be computed from previously computed $r_1^{(i)}$ and $r_2^{(i)}$ by restricting the bandwidth components to $(0, 1)$.

3.4 Algorithm

The formulae obtained in the previous Section motivate an algorithm for computing the selectivity of two-way joins selections using conjunctive equality predicates on base tables as well as the gradient of the error function with respect to the bandwidth parameter. We assume the number of distinct values for every attribute $|A|$ as given, as the number of distinct values per attribute is a common statistic in relational database systems. The algorithms are given in a Python-style pseudo code; Note that indentation indicates code blocks. We discuss generalizations to an arbitrary number of joins in Section 3.5.

3.4.1 Estimation

The selectivity estimation algorithm consists of five steps. The first three steps are given in Listing 3.1.

Step 1 (Line 5 - 8): We need to establish the domain equality assumption. To ensure that the selectivity stays in $[0,1]$ in case the assumption is violated, we set the number of distinct elements on the join attribute to the larger one.

Step 2 (Line 11 - 15): We sort both samples on the join attribute. Every sample additionally stores the attribute it is sorted on. We can therefore save computational effort in case one of the samples is sorted already.

Step 3 (Line 18 - 36): We scan the first sample S_1. We compute the invariant sample point contributions $r_1^{(i)}$ for every sample point and compute Ω_1 by incrementally summing them up and scaling with $\frac{\lambda_1}{|A|-1}$. The same steps are performed for the sample S_2, $r_2^{(i)}$ and Ω_2.

The remaining two steps are given in Listing 3.2:

Step 4 (Line 5 - 33): We use a merge algorithm on the join attribute and the previously computed invariant contributions. The algorithm groups the sample points in both relations by the join key, performs an aggregation of the invariant contributions and computes the unnormalized estimated probabilities $v1$ and $v2$ based on Ω_1 and Ω_2. These probabilities are then merged by normalizing, multiplying for all join keys v and adding to the overall selectivity estimate p. In case only one of the samples contains a particular join key, the samples are merged directly with the other samples Ω value as this value already represents the unnormalized estimate for join keys that are not part of the respective sample. The merging is sketched in Figure 3.3. Additionally, the algorithm counts the number of distinct handled values $dvals$, which is the quantity $|S_1.A_1 \cup S_2.A_2|$.

Step 5 (Line 35): The algorithm adds the contribution for all $|A| - |S_1.A_1 \cup S_2.A_2|$ values that are not in either of the samples by adding the join estimate for all join keys in neither of the samples.

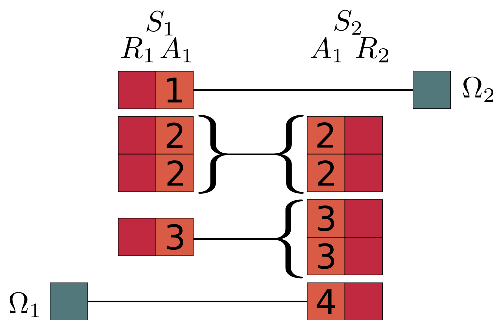


Figure 3.3: Merging step in the estimation algorithm

In case the samples are sorted, executing the algorithm requires only two passes over the sample. Hence, the runtime is linear in the sample size. Otherwise, we have to execute a sorting algorithm with a runtime complexity in $O(|S|log|S|)$ for every sample S.

Listing 3.1: Pseudocode for Step 1 to 3 of the join selectivity estimation algorithm

```
$r_1$ = [1,...,1]
$r_2$ = [1,...,1]
$\Omega_1$,$\Omega_2$ = 0.0

if $|R_1.A_1|$ > $|R_2.A_1|$:
    $|A|$ = $|R_1.A_1|$
else:
    $|A|$ = $|R_2.A_1|$

#Sort samples if necessary
if not $S_1$.sortedOn($A_1$):
    $S_1$.sort_on($A_1$)

if not $S_2$.sortedOn($A_1$):
    $S_2$.sort_on($A_1$)

#Calculate Omega values
for $\vec{s}^{(i)}$ in $S_1$:
    $r_1^{(i)}$ = 1.0
    for $(A_j = x_j)$ in $\sigma_1$:
        if $\vec{s}_j^{(i)}$ == $x_j$:
            $r_1^{(i)}$ *= $(1 - \lambda_j)$
        else:
            $r_1^{(i)}$ *= $\lambda_j$ / $(|R_1.A_j| - 1)$
    $\Omega_1$ += $r_1^{(i)}$
$\Omega_1$ *= $\lambda_1$/(|A|)

for $\vec{t}^{(i)}$ in $S_2$:
    $r_2^{(i)}$ = 1.0
    for $(A_j = x_j)$ in $\sigma_2$:
        if $\vec{t}_j^{(i)}$ == $x_j$:
            $r_2^{(i)}$ *= $(1 - \gamma_j)$
        else:
            $r_2^{(i)}$ *= $\gamma_j$ / $(|R_2.A_j| - 1)$
    $\Omega_2$ += $r_2^{(i)}$
$\Omega_2$ *= $\gamma_1$/(|A|)
```

Listing 3.2: Pseudocode for Step 4 and 5 of the join selectivity estimation algorithm

```
#Perform merge-style computation of the outer sum and distinct values
p = 0.0
dvals = 0

i,j = 1
if $s_1^{(i)} \leq t_1^{(j)}$:
    cur = $s_1^{(i)}$
else:
    cur = $t_1^{(j)}$

while i $\leq$ n or j $\leq$ m:
    dvals += 1
    v1 = $\Omega_1$
    v2 = $\Omega_2$
    while i $\leq$ m and $s_1^{(i)}$ == cur:
        v1 += $(1.0 - \lambda_1 - \frac{\lambda_1}{|A|-1}) \cdot (r_1^{(i)})$
        i += 1

    while j $\leq$ n and $t_1^{(j)}$ == cur:
        v2 += $(1.0 - \gamma_1 - \frac{\gamma_1}{|A|-1}) \cdot (r_2^{(j)})$
        j += 1

    p += v1 \cdot v2/(m \cdot n)

    if i $\leq$ m and j > n:
        cur = $s_1^{(i)}$
    elif i > m and j $\leq$ n:
        cur = $t_1^{(j)}$
    else:
        if $s_1^{(i)} \leq t_1^{(j)}$:
            cur = $s_1^{(i)}$
        else:
            cur = $t_1^{(j)}$

p += $\frac{|A|-\text{dvals}}{m \cdot n} \cdot \Omega_1 \cdot \Omega_2$
```

3.4.2 Error-Driven Bandwidth Optimization

Like with continuous Kernel Density Estimators, we can optimize the bandwidth parameter based on query feedback as all our previous results were continuously differentiable with respect to the bandwidth. For this thesis, we chose the batch-optimization approach, as it provided the best overall results in our previous work and is therefore best suited to assess the potential of our approach. First, we collect a set of representative queries with their true selectivities. During the collection, the estimator can already provide estimates by setting the bandwidth close to zero, which yields a simple sample estimator. Second, we use the constrained gradient-based optimization algorithm *MMA* [47] to optimize the bandwidth for a bandwidth vector $\vec{\lambda}$ with respect to a loss function $\mathcal{L}(x, y)$ on the estimated selectivity x and the true selectivity y. Formally, for a set of queries $Q = \{q_1, ..., q_n\}$, we solve the following optimization problem:

$$\underset{\vec{\lambda}}{\operatorname{argmin}} \sum_{q_i \in Q} \mathcal{L}\left(\hat{p}_{\vec{\lambda}}\left(q_i\right), p\left(q_i\right)\right) \tag{3.24}$$

The gradient of the objective with respect to a bandwidth vector $\vec{\lambda}$ is given by the sum of the gradient for every query:

$$\frac{\partial}{\partial \vec{\lambda}} \mathcal{L}\left(\hat{p}_{\vec{\lambda}}\left(q_i\right), p\left(q_i\right)\right) = \sum_{q_i \in Q} \frac{\partial}{\partial \vec{\lambda}} \mathcal{L}\left(\hat{p}_{\vec{\lambda}}\left(q_i\right), p\left(q_i\right)\right) \tag{3.25}$$

Using the chain rule yields the following derivative for $\mathcal{L}$:

$$\frac{\partial}{\partial \vec{\lambda}} \mathcal{L}\left(\hat{p}_{\vec{\lambda}}, \left(q_i\right), p\left(q_i\right)\right) = \frac{\partial}{\partial \hat{p}_{\vec{\lambda}}\left(q_i\right)} \mathcal{L}\left(\hat{p}_{\vec{\lambda}}\left(q_i\right), p\left(q_i\right)\right) \cdot \frac{\partial}{\partial \vec{\lambda}} \hat{p}_{\vec{\lambda}}\left(q_i\right) \tag{3.26}$$

The gradient of the loss function with respect to the estimate is independent of the actual query and can even be computed before the database finished query execution. While any continuously differentiable loss function suffices, we chose the quadratic error $L_{\text{Quadratic}}$ for this thesis, mainly because of good experiences in our previous publication [21].

$$\begin{aligned} \mathcal{L}_{\text{Quadratic}}\left(\hat{p}_{\vec{\lambda}}\left(q_i\right), p\left(q_i\right)\right) &= \left(\hat{p}_{\vec{\lambda}}\left(q_i\right) - p\left(q_i\right)\right)^2 \\ \frac{\partial}{\partial \hat{p}_{\vec{\lambda}}\left(q_i\right)} \mathcal{L}_{\text{Quadratic}}\left(\hat{p}_{\vec{\lambda}}\left(q_i\right), p\left(q_i\right)\right) &= 2 \cdot \left(\hat{p}_{\vec{\lambda}}\left(q_i\right) - p\left(q_i\right)\right) \end{aligned} \tag{3.27}$$

The queries in Q can consist of all operators supported by the kernels on the attributes. In case of models using only the categorical kernel, the queries can consist of selections with equality predicates and joins. If the query only consists of selections, the computation of the error-independent gradient of the estimator with respect to the bandwidth vector can be computed incrementally in one single pass following Equation 3.8. In case the query includes joins, we have to compute the more sophisticated gradient given in Equations 3.18 to 3.23. We provide a five step algorithm that exhibits a structure similar to the estimation algorithm. The first three steps are given in 3.3.

Step 1 (Line 6 - 9): We need to establish the the domain equality assumption. To ensure the selectivity stays in $[0, 1]$ in case the assumption is violated, we set the number of distinct elements on the join attribute to the larger one.

Step 2 (Line 12 - 16): We sort both samples on the join attribute. Every sample additionally stores the attribute it is sorted on. We can therefore save computational effort in case one of the samples is sorted already.

Step 3 (Line 19 - 39): We scan the first sample S_1. We compute the gradient of the invariant sample point contributions $\frac{\partial}{\partial \vec{\lambda}} r_1^{(i)}$ for every sample point and compute $\frac{\partial}{\partial \vec{\lambda}} \Omega_1$. To do so, we reuse $r_1^{(i)}$ and Ω_1, which are computed in the estimation algorithm. In case these quantities are not available, they can be re-computed in the same pass over the sample. The same steps are performed for the sample S_2, $\frac{\partial}{\partial \vec{\gamma}} r_2^{(i)}$ and $\frac{\partial}{\partial \vec{\gamma}} \Omega_2$.

The remaining two-steps are given in 3.4.

Step 4 (Line 5 - 41): We use a merge algorithm on the join attribute and the previously computed gradients to compute the gradient contributions for every value in $S_1 \cup S_2$.

Step 5 (Line 43 - 44): We add the gradient contributions for all values that are not in $S_1 \cup S_2$.

Like the estimation algorithm, the gradient algorithm requires two passes over the sample plus the cost of up to two sorts in case the samples were not sorted on the join key already. Note that the scan and merge phases of the estimation and gradient algorithm can be combined such that only two passes over the sample are necessary to compute both quantities.

Listing 3.3: Pseudocode for Step 1 to 3 of the gradient algorithm for join selectivities

```
$\frac{\partial}{\partial\vec{\lambda}}r_1$ = [[0,...,0],...,[0,...,0]]
$\frac{\partial}{\partial\vec{\gamma}}r_2$ = [[0,...,0],...,[0,...,0]]
$\frac{\partial}{\partial\vec{\lambda}}\Omega_1$ = [0,..0]
$\frac{\partial}{\partial\vec{\gamma}}\Omega_2$ = [0,..0]

if $|R_1.A_1|$ > $|R_2.A_1|$:
    $|A|$ = $|R_1.A_1|$
else:
    $|A|$ = $|R_2.A_1|$

#Sort samples if necessary
if not $S_1$.sortedOn($A_1$):
    $S_1$.sort_on($A_1$)

if not $S_2$.sortedOn($A_1$):
    $S_2$.sort_on($A_1$)


for $\vec{s}^{(i)}$ in $S_1$:
    for $(A_j = x_j)$ in $\sigma_1$:
        if $\vec{s}_j^{(i)}$ == $x_j$:
            $\frac{\partial}{\partial\lambda_j}r_1^{(i)}$ = $r_1^{(i)}$ / $-(1-\lambda_j)$
        else:
            $\frac{\partial}{\partial\lambda_j}r_1^{(i)}$ = $r_1^{(i)}$ / $\lambda_j$
        $\frac{\partial}{\partial\lambda_j}\Omega_1$ += $\frac{\partial}{\partial\lambda_j}r_1^{(i)}$
$\frac{\partial}{\partial\lambda_1}\Omega_1$ = $\Omega_1/\lambda_1$
for $(A_j = x_j)$ in $\sigma_1$:
    $\frac{\partial}{\partial\lambda_j}\Omega_1$ *= $\lambda_1$ / (|A|$-1$)

for $\vec{t}^{(i)}$ in $S_2$:
    for $(A_j = x_j)$ in $\sigma_2$:
        if $\vec{t}_j^{(i)}$ == $x_j$:
            $\frac{\partial}{\partial\gamma_j}r_2^{(i)}$ = $r_2^{(i)}$ / $-(1-\gamma_j)$
        else:
            $\frac{\partial}{\partial\gamma_j}r_2^{(i)}$ = $r_2^{(i)}$ / $\gamma_j$
        $\frac{\partial}{\partial\gamma_j}\Omega_2$ += $\frac{\partial}{\partial\gamma_j}r_2^{(i)}$
$\frac{\partial}{\partial\gamma_1}\Omega_2$ = $\Omega_2/\gamma_1$
for $(A_j = x_j)$ in $\sigma_2$:
    $\frac{\partial}{\partial\gamma_j}\Omega_2$ *= $\gamma_1$ / (|A|$-1$)
```

Listing 3.4: Pseudocode for Step 4 and 5 of the gradient algorithm for join selectivities

```
$\frac{\partial}{\partial\vec{\lambda}}p$ = [0,...,0]
$\frac{\partial}{\partial\vec{\gamma}}p$ = [0,...,0]
dvals = 0

i,j = 1
if $s_1^{(i)} \leq t_1^{(j)}$:
    cur = $s_1^{(i)}$
else:
    cur = $t_1^{(j)}$

while i $\leq$ m or j $\leq$ n:
    dvals += 1
    v1 = $\Omega_1$
    $\vec{\kappa}$ = $\frac{\partial}{\partial\vec{\lambda}}\Omega_1$
    while i $\leq$ m and $s_1^{(i)}$ == cur:
        for $(A_l = x_l)$ in $\sigma_1$:
            $\kappa_l$ += $(1.0 - \lambda_1 - \frac{\lambda_1}{|A|-1}) \cdot \frac{\partial}{\partial\lambda_l} r_1^{(i)}$
        $\kappa_1$ += $(-1.0 - \frac{1.0}{|A|-1}) \cdot r_1^{(i)}$
        v1 += $(1.0 - \lambda_1 - \frac{\lambda_1}{|A|-1}) \cdot r_1^{(i)}$
        i += 1

    v2 = $\Omega_2$
    $\vec{\vartheta}$ = $\frac{\partial}{\partial\vec{\gamma}}\Omega_2$
    while j $\leq$ n and $t_1^{(j)}$ == cur:
        for $(A_l = x_l)$ in $\sigma_2$:
            $\vartheta_l$ += $(1.0 - \gamma_1 - \frac{\gamma_1}{|A|-1}) \cdot \frac{\partial}{\partial\gamma_l} r_2^{(j)}$
        $\vartheta_1$ += $(-1.0 - \frac{1.0}{|A|-1}) \cdot r_2^{(j)}$
        v2 += $(1.0 - \gamma_1 - \frac{\gamma_1}{|A|-1}) \cdot r_2^{(j)}$
        j += 1

    $\frac{\partial}{\partial\vec{\lambda}}p$ += $\frac{1}{m \cdot n}\vec{\kappa} \cdot \mathrm{v2}$
    $\frac{\partial}{\partial\vec{\gamma}}p$ += $\frac{1}{m \cdot n}\mathrm{v1} \cdot \vec{\vartheta}$
    if i $\leq$ m and j > n:
        cur = $s_1^{(i)}$
    elif i > m and j $\leq$ n:
        cur = $t_1^{(j)}$
    else:
        if $s_1^{(i)} \leq t_1^{(j)}$:
            cur = $s_1^{(i)}$
        else:
            cur = $t_1^{(j)}$

$\frac{\partial}{\partial\vec{\lambda}}p$ += $\frac{|A|-\mathrm{dvals}}{m \cdot n} \cdot (\frac{\partial}{\partial\vec{\lambda}}\Omega_1) \cdot \Omega_2$
$\frac{\partial}{\partial\vec{\gamma}}p$ += $\frac{|A|-\mathrm{dvals}}{m \cdot n} \cdot \Omega_1 \cdot (\frac{\partial}{\partial\vec{\gamma}}\Omega_2)$
```

3.4.3 GPU-Implementation

Estimation and gradient computation for selections can be easily implemented on a GPU by computing the individual contribution of sample points in parallel and computing the sum using a binary reduction scheme. Except for the different formulae for estimation and gradient, all mechanisms introduced in Section 2.3.1 can be used for the categorical kernel as well.

However, adapting the join selectivity estimation algorithm for GPUs is a more complex task. Sorting the sample on the GPU can be done using GPU sorting algorithms like Bitonic-Merge Sort [37]. To sketch how the algorithm can be implemented on a GPU, we slightly modify Equation 3.17 to highlight intermediate results computed during the algorithm.

$$\begin{aligned}
\hat{p}\left(q\right) = & \sum_{v \in (S_1.A_1 \cup S_2.A_1)} \left(\frac{1}{m \cdot n} \left(\Omega_1 + \left(1 - \lambda_1 - \frac{\lambda_1}{|A|-1}\right) \sum_{i: s_1^{(i)} = v} r_1^{(i)} \right) \right. \\
& \left. \underbrace{\left(\Omega_2 + \left(1 - \gamma_1 - \frac{\gamma_1}{|A|-1}\right) \sum_{i: t_1^{(i)} = v} r_2^{(i)} \right)}_{c_2(v)} \right) + \frac{\left|A \backslash (S_1.A_1 \cup S_2.A_1)\right|}{m \cdot n} \cdot \Omega_1 \cdot \Omega_2 \\
= & \frac{1}{m \cdot n} \sum_{v \in (S_1.A_1 \cup S_2.A_1)} \underbrace{\left(\Omega_1 c_2\left(v\right) + \left(1 - \lambda_1 - \frac{\lambda_1}{|A|-1}\right) \sum_{i: s_1^{(i)} = v} r_1^{(i)} c_2\left(v\right) \right)}_{c(v)} \\
& + \frac{\left|A \backslash (S_1.A_1 \cup S_2.A_1)\right|}{m \cdot n} \cdot \Omega_1 \cdot \Omega_2 \\
= & \frac{1}{m \cdot n} \sum_{v \in (S_1.A_1 \cup S_2.A_1)} c\left(v\right) + \frac{\left|A \backslash (S_1.A_1 \cup S_2.A_1)\right|}{m \cdot n} \cdot \Omega_1 \cdot \Omega_2
\end{aligned} \tag{3.28}$$

The quantities $r_1^{(i)}$ and $r_2^{(i)}$ can be computed independently by different GPU threads for every sample point. The quantities Ω_1 and Ω_2 can be computed by summing over the previous values using a binary reduction scheme. This corresponds to the scan of the sample in the sequential algorithm. Figure 3.4 schematically shows the steps of the algorithm that lead to the computation of Ω_1 for a toy example.

Computing the product of the per-sample contributions $c(v)$ for every value $v \in S_1.A_1 \cup S_2.A_1$ corresponds to the merge step in the sequential algorithm. For a GPU implementation, we suggest an algorithm that resembles a binary search join [19]. We select one relation as the probe relation - assume it is S_2. We have one thread $t_x, x \in \{1, |S_1|\}$ for each tuple in S_1.

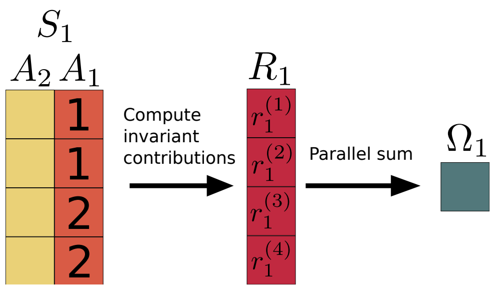


Figure 3.4: Computation of invariant contributions r and Ω in the GPU algorithm for a toy example

Every thread performs a binary search in S_2 for matching join keys. Given v_x being the join key of tuple $\vec{s}_x$, every thread t_x incrementally computes $c_2(v_x)$ by visiting matching tuples and stores the value at position $x-1$ a buffer C_1. In case a thread t_x does not find at least one qualifying tuple in S_2, it sets $c_2(v_x) = \Omega_2$, which is consistent with the definition of c_2. Using this technique, the algorithm can compute $c_2(v)$ for all values $v \in S_1.A_1$. However, values that are in $S_2.A_1 \setminus S_1.A_1$ have to be treated as well to cover all values $v \in S_2.A_1 \cup S_1.A_1$. To do so, every tuple in S_2 is assigned a flag in a buffer F, that is initially zero for all tuples. Whenever a thread v_x finds a matching tuple $\vec{s}_x$ during the binary search it sets the corresponding flag in F, signaling that the join key is also included in S_1.

Figure 3.5 shows the state of the algorithm for the toy example after performing the binary search step with probe sample S_2. The arrows point from sample points that have been found during the binary search to the positions in the buffer C_1 they contributed to. Every tuple in S_2 with an originating arrow has its flag in F set.

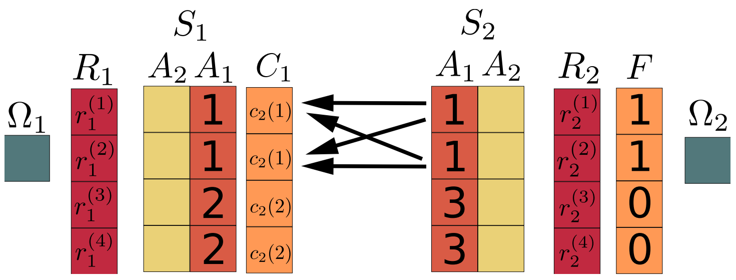


Figure 3.5: State of the GPU algorithm after performing the binary search step for a toy example

After these computations, we need to perform one further aggregation step to compute $c(v)$ for all values $v \in S_1.A_1$. For all values v in $S_1.A_1$, we basically need to compute $\sum_{i:s_1^{(i)}=v} r_1^{(i)} c_2(v)$, the remaining operations can be applied in a finalizing step of this computation. We implement this computation by performing atomic add operations to an output buffer.

To do so, every tuple with a join key v needs to perform an atomic add to the same position in the output array. We use a common pattern in GPU programming to compute these positions that is based on evaluating a binary predicate and computing a prefix sum [19, 37].

To compute the predicate, every thread $t_x, x \in \{1, |S_1|\}$ sets its position in a zero initialized buffer to one, whenever $\vec{s}^{(x)}$ is the first tuple in the sorted sample that has join key v_x. Afterwards, a prefix sum with an initial value of -1 can be computed from this buffer to assign a unique position in an output array to every v. As the samples are sorted on the join key, tuples with the same join key are assigned the same value by the prefix sum. Furthermore, the prefix sum allows us to determine the number of distinct keys in S_1.

We apply similar computations to compute $c(v)$ for all $v \in S_2.A_1 \setminus S_1.A_1$. As these join keys v have no matching join key in R_1, the computation of $c(v)$ reduces to $\Omega_1 \cdot c_2(v)$. All tuples with such keys have been identified in the binary search step by leaving their flag in F unset. The quantity $c_2(v)$ for these values can be computed using atomic add operations as well and all threads responsible for tuples with the same join key $v \in S_2.A_1 \setminus S_1.A_1$, have to add to the same position in an output buffer. We evaluate the same predicate for every tuple in S_2, but additionally require an unset flag in F to evaluate to 1. Afterwards, we apply the prefix sum.

Figure 3.6 shows the results of the predicate and prefix sum computations for both samples of our toy example. For S_1, the prefix sum assigns the position 0 to join key 1 and position 1 to join key 2. Note that the last entry of the prefix sum plus one yields $|S_1.A_1|$. For S_2 the modified prefix sum assigns position -1 to join key 1 as the predicate evaluated to 0. Tuples with this join key are covered already by S_1 and, thus, do have their flag in F set. For join key 3 position 0 is assigned subsequently. Note that the last entry of the prefix sum plus one yields $|S_2.A_1 \setminus S_1.A_1|$.

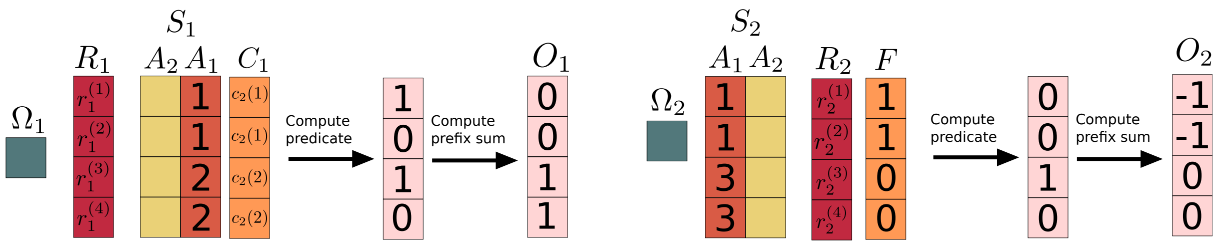


Figure 3.6: Computation of predicates and prefix sums in the GPU algorithm for a toy example

Now, every thread $t_x, x \in \{1, |S_1|\}$ can compute $r_1^{(x)} \cdot c_2(v)$ and use an atomic add operation to its position in the output array, a finalizing step adds Ω_1 and scales the aggregates to yield $c(v)$. Similarly, we compute $c(v)$ for $v \in S_2.A_1 \setminus S_1.A_1$. However, only the t_x with an unset flag in F participate in the aggregation. Note that in current GPUs atomic add operations are only supported for 32-bit floating point types [35].

Figure 3.7 shows the application of this step. For every sample, each entry in the respective prefix sum that is non-negative corresponds to a position in the output array.

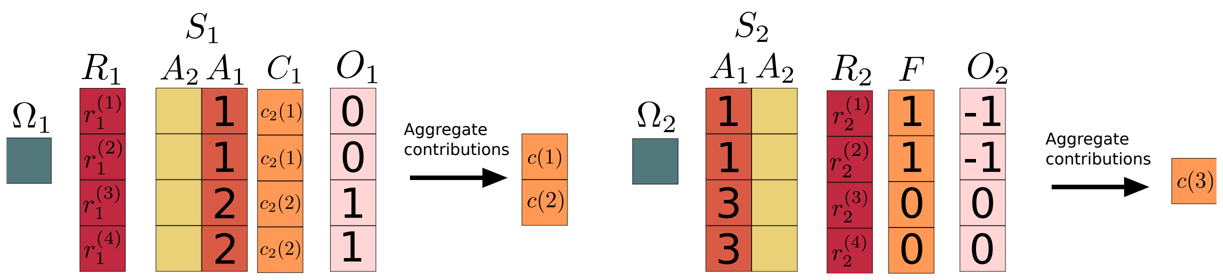


Figure 3.7: Computation of $c(v)$ in the GPU algorithm for a toy example

After these computations, all quantities required for the estimate computation are readily available and can be computed in a last finalizing step that sums all $c(v)$ values and performs the remaining trivial computations.

Computing the gradient can be implemented in a similar fashion as estimation and gradient computation follow the same structure.

3.4.4 Sample Maintenance

The sample can be maintained by the exact or heuristic algorithms for sample maintenance. An overview of possible algorithms is given in [28]. As our algorithm for join selectivity estimation requires a sample sorted on a join key, sample points replaced by a maintenance algorithm may break an existing sort order. Either the new sample point has to be inserted order preserving or the sample has to be flagged unordered to trigger re-sorting at the next arriving estimation request. Ordered insertion is preferable when the majority of requests deal with the same join attribute and an existing order can be reused. Resorting the entire sample is preferable when join attributes change frequently.

3.5 Generalization for n-Way Joins

The equalities and algorithms in the previous two Sections can be extended to n-way joins including a total of n equi-join operators. We revisit the two remaining cases in Section 3.2. If one of the join attributes in an additional join is one of the previously joined columns, extending the algorithm is simple. Again, we substitute the true selectivity p for the estimate provided by Kernel Density Estimator and restrict the selections to equality predicates.

$$\begin{aligned} &q_3 : \sigma_1(R_1) \bowtie_{R_1.A_1=R_2.A_1} \sigma_2(R_2) \bowtie_{R_2.A_1=R_3.A_1} \sigma_3(R_3) \\ &\hat{p}(q_3) = \sum_{v \in R_1.A_1 \cap R_2.A_1 \cap R_3.A_1} \hat{p}_{R_1}(A_1 = v \wedge \sigma_1)\, \hat{p}_{R_2}(A_1 = v \wedge \sigma_2)\, \hat{p}_{R_3}(A_1 = v \wedge \sigma_3) \end{aligned} \tag{3.29}$$

This simply extends the existing case by an additional relation and does not introduce further complexity. We assume $|A| = |R_1.A_1| = |R_2.A_1| = |R_3.A_1|$ and apply the same transformations given in Section 3.3. The estimation and gradient algorithm sort, scan and merge the third sample together with the other ones.

Unfortunately, in case the join attributes do not match a previously joined attribute, additional complexity is introduced as a Cartesian product of the join keys has to be considered.

$$\begin{aligned} &q_4 : \sigma_1(R_1) \bowtie_{R_1.A_1=R_2.A_1} \sigma_2(R_2) \bowtie_{R_2.A_2=R_3.A_1} \sigma_3(R_3) \\ &\hat{p}(q_4) = \sum_{\substack{v_1 \in R_1.A_1 \cap R_2.A_1; \\ v_2 \in R_2.A_2 \cap R_3.A_1}} \hat{p}_{R_1}(A_1 = v_1 \wedge \sigma_1)\, \hat{p}_{R_2}(A_1 = v_1 \wedge A_2 = v_2 \wedge \sigma_2)\, \hat{p}_{R_3}(A_1 = v_2 \wedge \sigma_3) \end{aligned} \tag{3.30}$$

A possible approach would be computing $\hat{p}_{R_1}(A_1 = v_1 \wedge \sigma_1)$ for all values v_1 and starting an instance of the two-way join algorithm for the remaining two relations with $A_1 = v_1$ fixed by a selection. However, this approach does not scale as the number of estimator evaluations depends on the domains of the attributes. Making an independence assumption for relations with multiple join attributes, allows us to maintain the scalability of the algorithm. We partition the attributes participating in the selection σ_2 into two disjoint selections $\sigma_{2.1}$ and $\sigma_{2.2}$ which are assigned to either one of the join attributes and considered independent. For this thesis, we partition the selections evenly based on their position in the relation schema.

$$\hat{p}_{R_2}(A_1 = v_1 \wedge A_2 = v_2 \wedge \sigma_2) = \hat{p}_{R_2}(A_1 = v_1 \wedge \sigma_{2.1})\, \hat{p}_{R_2}(A_2 = v_2 \wedge \sigma_{2.2}) \tag{3.31}$$

This allows the following transformations:

$$\begin{aligned} &\hat{p}(q_4) = \\ &\quad \sum_{\substack{v_1 \in R_1.A_1 \cap R_2.A_1; \\ v_2 \in R_2.A_2 \cap R_3.A_1}} \hat{p}_{R_1}(A_1 = v_1 \wedge \sigma_1)\, \hat{p}_{R_2}(A_1 = v_1 \wedge \sigma_{2.1})\, \hat{p}_{R_2}(A_2 = v_2 \wedge \sigma_{2.2})\, \hat{p}_{R_3}(A_1 = v_2 \wedge \sigma_3) \\ &\quad = \left(\sum_{v_1 \in R_1.A_1 \cap R_2.A_1} \hat{p}_{R_1}(A_1 = v_1 \wedge \sigma_1)\, \hat{p}_{R_2}(A_1 = v_1 \wedge \sigma_{2.1}) \right) \\ &\quad \cdot \left(\sum_{v_2 \in R_2.A_2 \cap R_3.A_1} \hat{p}_{R_2}(A_2 = v_2 \wedge \sigma_{2.2})\, \hat{p}_{R_3}(A_1 = v_2 \wedge \sigma_3) \right) \end{aligned} \tag{3.32}$$

This quantity and its gradient can be computed by running two instances of the introduced algorithms and multiplying their estimates.

We can estimate arbitrary equi-joins with equality selection predicates by applying the previously introduced changes to the algorithms.

3.6 Discussion

We discuss properties and drawbacks of our estimators in this Section. Most importantly, Kernel Density Estimator models using the categorical kernel can always be parameterized to yield the relative frequency distribution in the sample by setting the components of the bandwidth vector to zero. This is the case, as the categorical kernel always places contributes $1 - \lambda$ whenever sample and function value coincide. Thus, the contribution of a sample point is only one if it matches the function value entirely, given that all bandwidth components are zero. Hence, our estimators can always be turned into estimators that are entirely based on the evaluation of independent samples. This even works for our approach to join selectivity estimation for two-way joins and n-way joins with one join attribute per table - despite our assumption of equal value sets in join relations. Taking Equation 3.16 in Section 3.3 and setting $\vec{\lambda}$ and $\vec{\gamma}$ to zero-vectors, we obtain:

$$\begin{aligned}
\hat{p}(q) = & \sum_{v \in S_1.A_1 \cup S_2.A_1} \left(\frac{1}{m \cdot n} \left(\sum_{i=1}^{m} k_{\lambda_j}\left(s_j^{(i)}, v\right) r_1^{(i)} \right) \left(\sum_{i=1}^{n} k_{\gamma_j}\left(t_j^{(i)}, v\right) r_2^{(i)} \right) \right) \\
& + \frac{\left|A \backslash (S_1.A_1 \cup S_2.A_1)\right|}{m \cdot n} \left(\frac{\lambda_1}{|A|-1} \sum_{i=1}^{m} r_1^{(i)} \right) \left(\frac{\gamma_1}{|A|-1} \sum_{i=1}^{n} r_2^{(i)} \right) \\
\underset{\vec{\lambda}=\vec{0}, \vec{\gamma}=\vec{0}}{=} & \frac{1}{m \cdot n} \sum_{v \in S_1.A_1 \cup S_2.A_1} \left(\left(\sum_{i=1}^{m} k_{\lambda_j}\left(s_j^{(i)}, v\right) r_1^{(i)} \right) \left(\sum_{i=1}^{n} k_{\gamma_j}\left(t_j^{(i)}, v\right) r_2^{(i)} \right) \right) \\
= & \frac{1}{m \cdot n} \sum_{v \in S_1.A_1 \cup S_2.A_1} \left| q\left(S_1, S_2\right) \right|
\end{aligned} \tag{3.33}$$

As the bandwidth vector is the zero vector, $r_2^{(i)}$ and $r_2^{(j)}$ have a value of one, whenever their respective sample points fulfill the selection predicate, otherwise they have a value of zero. Similarly, $k_{\lambda_j}\left(s_j^{(i)}, v\right)$ and $k_{\gamma_j}\left(t_j^{(i)}, v\right)$ are one for all sample points with a value of v. Thus, the sums over the sample points compute the number of tuples with a value v that fulfill the selection predicates. Their product is the contribution of tuples with the value v to the join between both samples. The sum over all values in both samples is the output cardinality of the query evaluated on the samples, which, when divided by the product of the sample sizes, yields the simple sampling estimator introduced in Section 2.2.1.

Our algorithm forbids bandwidth components that are equal to zero, as this restriction eases the gradient computation. However, bandwidth components sufficiently close to zero practically have the same effect. As simple sample evaluation is part of the parameter space of the error-driven bandwidth optimizer, a properly trained KDE model should not perform worse than a simple sample based estimator under the optimized error function.

The assumption of equal value sets in join attributes made in our estimation algorithm is even stronger than the containment of value sets assumption in text-book selectivity estimation (Section 2.1. However, error-driven bandwidth optimization has the possibility to balance the impact of the assumption by choosing the bandwidth on the join attribute to place more probability mass on values that are included in the sample. In the most extreme case, the bandwidth on the join attribute can be chosen close to zero, which causes the estimator to place probability mass only on values that are included in the sample and guaranteed to be present in the relation (given there were no changes in the dataset). However, the assumption may introduce a problem to error-driven bandwidth optimization. While it is possible to optimize the bandwidth for a set of queries Q that consists of selections and bandwidth, we suggest to maintain and optimize different bandwidth vectors for selections and joins. In case our assumption of equal value domains on the join attribute is violated, the number of distinct values on an attribute is adjusted for one relation. Thus, the bandwidth parameters for selections and joins would not be computed based on the same model. Furthermore, as the errors in join size estimates tend to be much larger than for selections, the bandwidth is likely to be a value that is suboptimal for selections.

As Kernel Density Estimators are based on independent samples, it is an important question whether the categorical kernel can fix the problems of independent uniform samples. The severe drawback of independent samples is that they are not capable of modeling values that are not fully included in the sample. When it comes to join selectivity estimation for relational databases, this causes a problem: In case the number of distinct values in a join attribute is larger than the sample size and the values are uniformly distributed or at least not significantly skewed, samples need to be sufficiently large to have a reasonable chance of including matching join values. If there are no matching join attributes, the estimator will output zero as the join between two samples is empty. Kernel Density Estimator models using the categorical kernel are capable of providing estimates for values that are not or only partially included in the sample. Thus, the problem does not apply in the same way to our join selectivity estimator. However, our estimator is nevertheless limited in the way that it can deal with join keys that are not in the sample. Looking at Equation 3.16 again, a join value v that is not included in a sample S_1 always contributes the factor $\frac{\lambda_1}{|A|-1}\sum_{i=1}^{m} r_1^{(i)}$. While this is the key fact that allows us an efficient computation of join selectivity estimates, it also has important implications. Every value v not included in the sample will contribute the same probability, as the factor is entirely independent of v. Thus, for every value that is not included in a sample, we will not be able to provide estimates that capture the dependency between the join attribute and selection attributes. Intuitively, this can be seen as making an independence assumption of such values. In case $|A| >> n$ and a close to uniform distribution in A, the majority of values will be computed using this simple factor. Furthermore, a bandwidth selector will likely account for the unobserved values by choosing a bandwidth λ_1 close to one. Thus, we have $\frac{\lambda_1}{|A|-1} \sum_{i=1}^{m} r_1^{(i)} \approx \frac{1}{|A|} \sum_{i=1}^{m} r_1^{(i)}$, which is similar to making the uniformity assumption on the join attribute. Thus, in case the sample is too small to provide reasonable estimates by means of sample evaluation, our estimator falls back to estimates that rely on the independence and uniformity assumption. Thus, our estimator has inherited the problem of independent samples but provides a better worst case.

In the following Chapter, we evaluate our estimator experimentally and show that the discussed properties of our estimator hold.

Part IV

Evaluation & Conclusion

4 Evaluation

In this Section we perform an experimental evaluation of our proposed estimators. We evaluate our estimators in terms of estimation quality on a synthetic and a real-world dataset and compare them to other estimators to highlight their properties. Furthermore, we investigate the performance of our estimators in terms of execution time for estimate and gradient computations.

4.1 Experimental Setup

First, we cover base table selections with conjunctive equality predicates. We compare three estimators:

KDE: A discrete Kernel Density Estimator as described in Section 3.1.1.

Sample: A simple sample-based estimator for selections as described in Section 2.2.1. We draw a simple random sample S without replacement from the input relations, evaluate queries on the sample and scale the output cardinality by $\frac{1}{|S|}$.

Postgres: The estimator used by a vanilla installation of the PostgreSQL [39] database system (Version 9.5.0). We extract the estimates using the *ANALYZE* command. PostgreSQL makes the attribute independence assumption and obtains per-column selectivities by maintaining equi-width histograms, the frequencies of the most common values and the number of distinct values [38].

Thus, we can compare the estimates of Kernel Density Estimator models to simple samples, where we would expect *KDE* to provide equivalent or better estimates, and compare their estimates against a popular database system as a baseline.

Second, we cover join selectivities. We compare five estimators:

JoinKDE: The join selectivity estimator based on two discrete Kernel Density Estimator models as described in Section 3.3 to 3.5.

Sample: The simple sample based estimator described in Section 2.2.1.

AGMS: The AGMS Sketch with the extension to filter conditions as described in Section 2.2.2.

We use the *Polynomials over Primes* scheme [41] to construct the 4-wise independent hash functions.

Postgres: The estimator used by a vanilla installation of the PostgreSQL [39] database system (Version 9.5.0). We extract the estimates using the *ANALYZE* command. PostgreSQL makes the attribute independence assumption and obtains per-column selectivities from equi-width histograms, the frequencies of the most common values and the number of distinct values. Attribute uniformity is assumed on the join attribute. Except for the more detailed per-column statistics for selections, the computations are equivalent to the text-book approach given in Section 2.1 [30, 38].

KDE: A Kernel Density Estimator, as described in Section 3.1.1, that was constructed using a uniform sample of the join result. With this baseline, we can compare the previous estimators to a model that was constructed from the join result instead of using statistics on the base tables.

We compare the estimators on two different datasets.

IMDb: The IMDb dataset [23] is a plain text dump of the Internet Movie Database that collects and provides information on movies, TV programs and video games. We used the IMDbPY [24] Python script to import the dataset in PostgreSQL.

Zipf(s)*:* An artificial dataset that is based on the Zipfian distribution. Attribute *c1* is obtained by drawing 10000 values from a Zipfian distribution with parameter s. To limit the domain, we take the drawn values modulo 10000. The remaining attributes $ci, i \in \{2, 3, 4\}$ take the value of the previous attribute with probability $\frac{1}{2^{i-1}}$, otherwise they take a random value in $[0, 9999]$.

The first dataset is used to show the performance on real-world data, while the second dataset is used to show the behavior of the algorithm when it comes to skew and dependencies between attributes.

We evaluate the estimators for queries that consist of up to two joins and selections with equality predicates. We construct our queries by fixing a join attribute (if there is one) as well as selection attributes. Then, we draw a uniform sample of 400 tuples from the query result without applying selections, i.e. directly from a base table or join result. We use these tuples to construct 400 queries by parameterizing the selections with the tuple value on the respective attribute. We use 300 of these queries to train the bandwidth parameter of Kernel Density Estimator models. The bandwidth optimization was performed by an off-the-shelf implementation of the *MMA* algorithm [47] from the *NLopt* library [26] for nonlinear optimization. Afterwards, we use the remaining 100 queries for evaluation of all estimators. We report the average squared estimation error in tuples for 20 iterations of training and evaluation.

4.2 Model Size & Estimation Quality

In the following experiments we evaluate the behavior of the estimators for selections and joins with respect to estimation quality for varying model sizes. We assume every attribute in samples to be given as an integer. For *AGMS*, we account one integer for every sketch and every seed required for the hash function. Though our hash function requires four seeds, we chose to account only one seed as there exist more memory-efficient hashing schemes [41]. The reported model size is the number of tuples used for a single sample in the *Sample* estimator. All other estimators are set up to use the same amount of memory.

4.2.1 Selections

First, we evaluate the estimation quality of our estimator for base table selections only. We start with a set of queries on the IMDb dataset and show how estimates are effected by incrementally adding selections. Query set **IMDb QA** selects from the *title* table that contains 3.5 million tuples for every opus in the Internet Movie Database:
IMDb QA.1: Select all opuses that were produced in a given year.

```
select * from title where production_year = $x
```

IMDb QA.2: Select all opuses that were produced in a given year and are of a certain kind (for example TV series or game).

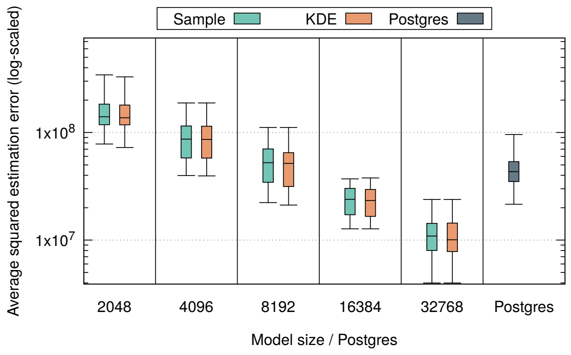
```
select * from title where production_year = $x and kind_id = $y
```

IMDb QA.3: Select all opuses that were produced in a given year, are of a given kind and have a given season number.

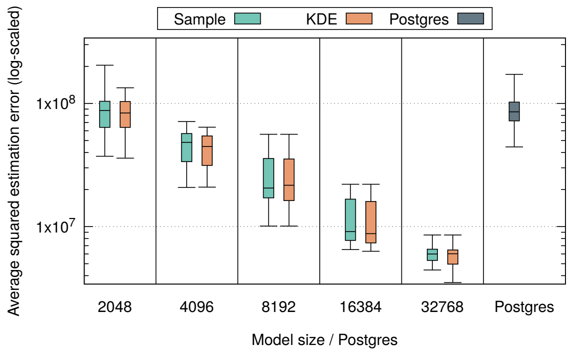
```
select * from title where production_year = $x
                    and kind_id = $y and season_nr = $z
```

Figure 4.1 shows boxplots for the average squared estimation error of these queries for *Postgres* as well as *Sample* and *KDE* with varying sample sizes. The errors for *Sample* and *KDE* barely differ. The error-driven bandwidth selector has chosen to the bandwidth close to zero rendering *KDE* and *Sample* equivalent. *Postgres* performs very well for selections on a single attribute (IMDb QA.1) and large samples are required to outperform it. However, when multiple selections are involved even small samples provide significantly better estimates as the independence assumption introduces errors. For example, much more video games were produced in years after the millennium than before.

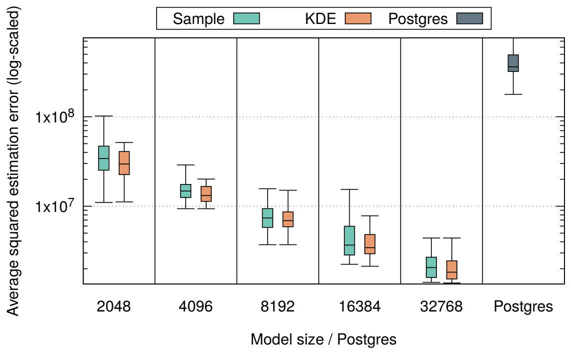
As the selections in the previous query have a high selectivity, we tested the estimator on the query set **IMDb QB** with more restrictive selections. It selects tuples from the *cast_info* table



(a) Estimation quality for query IMDb QA.1



(b) Estimation quality for query IMDb QA.2



(c) Estimation quality for query IMDb QA.3

Figure 4.1: Estimation quality for query set IMDb QA covering selections on base tables

that contains 50 million tuples linking items from the *title* table to their cast.

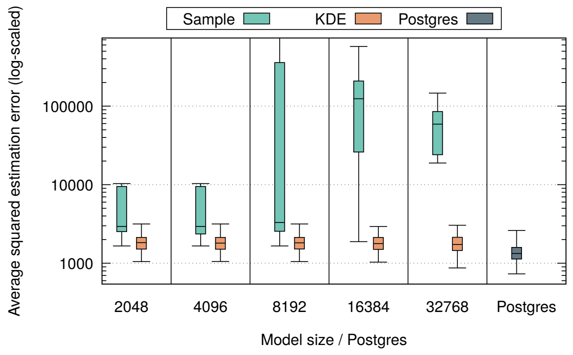
IMDb QB.1: Select all persons that are cast members of a given opus.

```
select * from cast_info where movie_id = $x
```

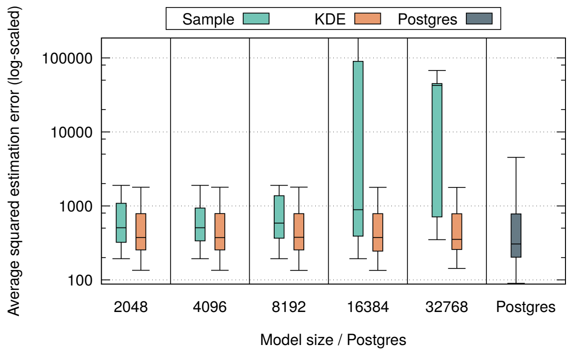
IMDb QB.2: Select all persons that are cast members of a given movie fulfilling a given role (e.g. actor, composer, director).

```
select * from cast_info where movie_id = $x and role_id = $y
```

Figure 4.2 shows the error plots for this query set. For these queries, *KDE* performs clearly better for most sample sizes. For both queries, we see the estimation quality of *Sample* deteriorating with the model size. While the loss in accuracy might seem counter-intuitive, it is easy to explain: *Sample* can only return non-zero estimates when at least one tuple matches the predicates. For small sample sizes the probability that one of the evaluation queries matches tuples in the sample is low due to the selection on the attribute *movie_id*, that contains thirty million different values. As the selectivity is very low, estimating a selectivity close to zero is a reasonable estimate. With an increasing sample size, tuples in the sample start matching the evaluation queries, however, scaling the result on the sample to the entire dataset causes overestimations. We see *Postgres* and *KDE* delivering very similar estimation quality for both



(a) Estimation quality for query IMDb QB.1



(b) Estimation quality for query IMDb QB.2

Figure 4.2: Estimation quality for query set IMDb QB covering selections on base tables

queries. The bandwidth selector chose to neglect the values in sample for the *movie_id* attribute, as the estimation error of *KDE* does not improve with the model size. As we have discussed in Section 3.6, the behavior of *KDE* is close to the uniformity and independence assumption in case an attribute is not skewed and the number of distinct elements exceeds the sample size by far. It appears that the estimates provided by Postgres are very close to a uniform distribution for *movie_keyword*, which explains the similar estimation quality in IMDb QB.1. For IMDb QB.2, both estimators seem to have a similarly accurate model of the distribution of *role_id* as well. As *KDE* is unable to consider attribute dependencies for the evaluated sample sizes, the observed estimation errors are very close.

We conducted a similar experiment using a relation *z1* that is an instance of *Zipf(*1.04*)* with query set **Zipf QA**: In the first query, we select on attribute *c1* In the remaining two queries, we add the partially dependent columns *c2* and *c3* to the selection.

Zipf QA.1: A single selection on attribute *c1* which has the highest skew in the data distribution.

```
select * from z11 where c1 = $x
```

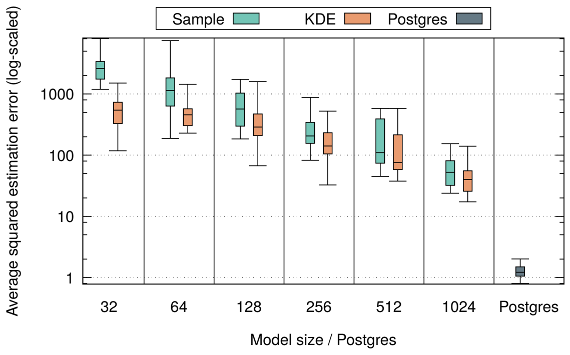
Zipf QA.2: We add the partial dependent column *c2* to the selection.

```
select * from z11 where c1 = $x and c2 = $y
```

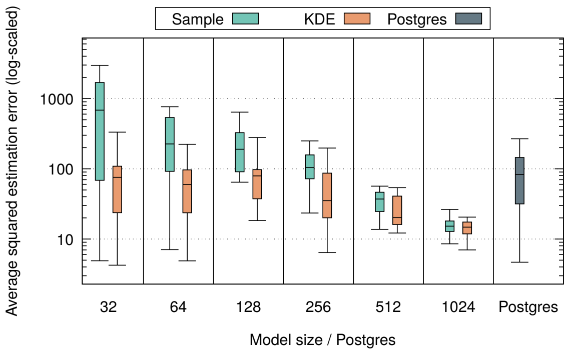
Zipf QA.3: We add the partially dependent column *c3* to the selection.

```
select * from z11 where c1 = $x and c2 = $y and c3 = $z
```

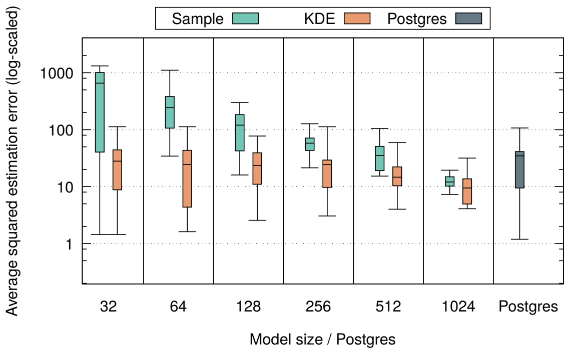
The results are shown in Figure 4.3. For Zipf QA.1, we find *Postgres* providing superior estimates. Column *c1* was generated from a Zipfian distribution. Thus, it contains a lot of values that occur once or twice, which are handled very well by a histogram, and few more frequent values, which are handled by maintaining the frequencies of most common values. However, the performance of *Postgres* deteriorates when dependent attributes are included. We



(a) Estimation quality for query Zipf QA.1



(b) Estimation quality for query Zipf QA.2



(c) Estimation quality for query Zipf QA.3

Figure 4.3: Estimation quality for query set Zipf QA covering selections on base tables

find *KDE* providing better estimates than *Sample* in all queries, in particular for small model sizes. This is due to the estimators ability to provide estimates for values that are not or only partially included in the sample as especially *c*2 and *c*3 contain a lot of distinct values.

The previous experiments give us two insights: First, the estimates provided by *Postgres* are not always inferior to *Sample* and *KDE*. In particular, *Postgres* provides reasonable estimates for selections on a single-attribute and selections with a very low selectivity. Second, *KDE* always provides significantly better or similar estimates when compared to *Sample*.

4.2.2 Joins

In this Section we cover queries including joins and selections. We evaluate queries including two-way joins and three-way joins. Furthermore, we investigate the effect of conflicting bandwidth values for joins and selections due to the assumption of equal value domains on join attributes.

4.2.2.1 Two-Way Joins

First, we evaluate the performance of our estimator on the IMDb dataset. We use query set **IMDb QC** containing the following queries:
IMDb QC.1: All entries in the *title* table with a given type and associated with a given keyword.

```
select * from movie_keyword,title where
              movie_keyword.movie_id = title.id
              and movie_keyword.keyword_id = $x
              and title.kind_id = $y
```

IMDb QC.2: All cast members that had a given role in a movie with a given keyword.

```
select * from movie_keyword,cast_info where
              movie_keyword.movie_id = cast_info.movie_id
              and movie_keyword.keyword_id = $x
              and cast_info.role_id = $y
```

IMDb QC.3: All companies of a given type operating in a given country.

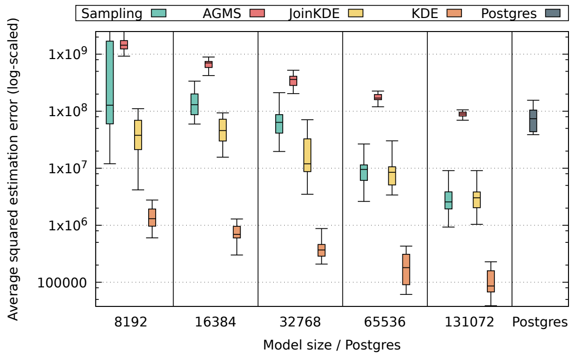
```
select * from movie_companies,company_name where
              movie_companies.company_id = company_name.id
              and movie_companies.company_type_id = $x
              and company_name.country_code = $y
```

IMDb QC.4: All cast members that have played a given role in movies from a given year.

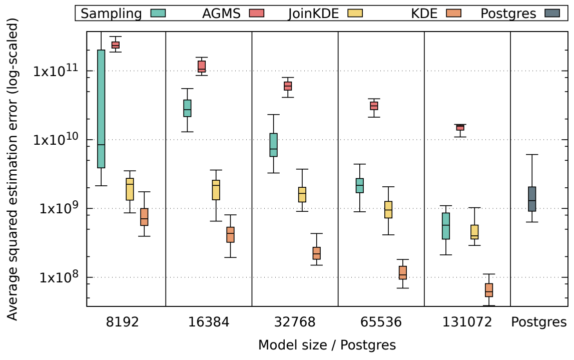
```
select * from cast_info,title where
              cast_info.movie_id = title.id
              and cast_info.role_id = $x
              and title.production_year = $y
```

IMDb QC.1 and IMDb QC.2 strongly violate the assumptions of *JoinKDE* as the *movie_keyword* table contains less than 20% of the keys contained in *cast_info* or *title*. The assumption does hold for IMDb QC.3 and is mildly violated for IMDb QC.4 with a value containment of 90%.

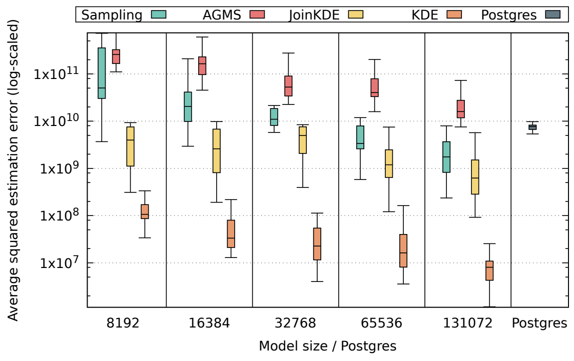
Figure 4.4 shows the error plots for the given queries. *KDE* clearly outperforms all other estimators and shows a good scaling behavior with the model size. This is expected behavior as we bypass the problems of independent samples by directly sampling from the join result. *AGMS* can barely compete with any of the other estimation techniques. *JoinKDE* outperforms *Sample* - especially for small model sizes. Interestingly, while the estimation quality of *Sample* improves with increasing sample size, *JoinKDE* only shows clear improvements for model sizes 32768 and larger. As *JoinKDE* clearly outperforms *Sample* for small model sizes, a significant amount of the estimation comes from kernels distributing probability mass on values not in the sample. Thus, increasing the sample size does not necessarily lead to better estimates for *JoinKDE*.



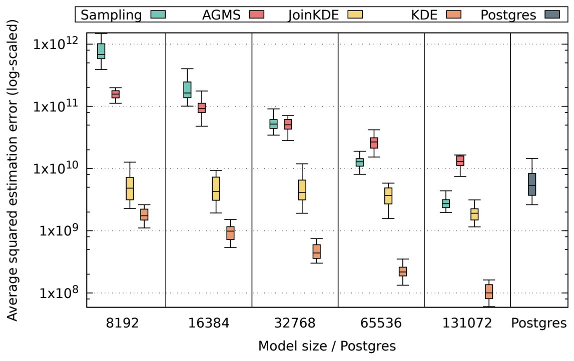
(a) Estimation quality for query IMDb QC.1



(b) Estimation quality for query IMDb QC.2



(c) Estimation quality for query IMDb QC.3



(d) Estimation quality for query IMDb QC.4

Figure 4.4: Estimation quality for query set IMDb QC covering 2-way joins subject to selections

When comparing *Join KDE* to *Postgres*, we notice that the estimates are comparable for smaller sample sizes. While *JoinKDE* may have accurate estimates on the selection attributes, we join on large tables and attributes that are either surrogate keys or foreign keys to such attributes that are barely skewed. This leads to low probabilities for having a particular join key in the samples. The error-driven bandwidth selector should pick a value close to one on these attributes, which leaves the estimator with a close to uniform distribution for the join attribute. Whenever a key is not included in the sample, the estimate is computed by multiplying the per attribute selectivities which is equivalent to making an independence assumption. As these are the assumptions made by *Postgres*, it is not surprising that the estimates only improve significantly for sufficiently sample sizes.

We further investigate the effect of skew in the join attributes as well as dependence between the join and selection attribute by using the artificial *Zipf* dataset. We use two tables *z11* and *z12*, each being an instance of $Zipf(1.04)$, as well as two tables *z21* and *z22*, each being an instance of of $Zipf(1.005)$. Note that a higher Zipfian parameter results in more skew in the dataset. We use these tables to create query set **Zipf QB**:

Zipf QB.1: Join the tables with higher skew on attribute *c1* and select on the highly dependent attribute *c2*.

```
select * from z11,z12 where
              z11.c1 = z12.c1
              and z11.c2 = $x
              and z12.c2 = $y
```

Zipf QB.2: Join the tables with higher skew on attribute *c1* and select on the less dependent attribute *c4*.

```
select * from z11,z12 where
              z11.c1 = z12.c1
              and z11.c4 = $x
              and z12.c4 = $y
```

Zipf QB.3: Join the tables with lower skew on attribute *c1* and select on the highly dependent attribute *c2*.

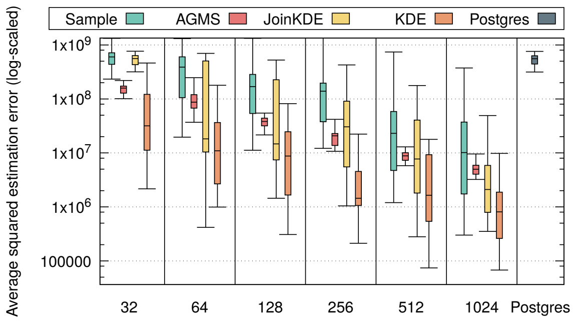
```
select * from z21,z12 where
              z21.c1 = z22.c1
              and z21.c2 = $x
              and z22.c2 = $y
```

Zipf QB.4: Join the tables with lower skew on attribute *c1* and select on the less dependent attribute *c4*.

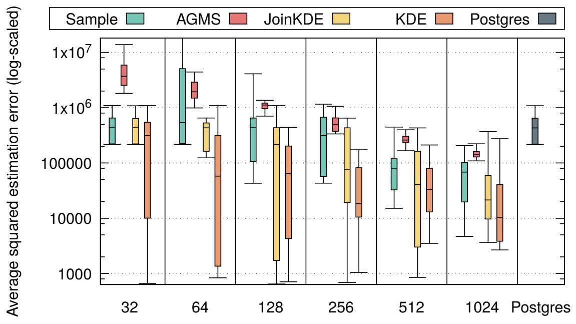
```
select * from z21,22 where
              z21.c1 = z22.c1
              and z21.c4 = $x
              and z22.c4 = $y
```

These queries are close to self-joins as they are distributed very similarly. As for the previous query set, *KDE* clearly dominates for almost all combinations of queries and model sizes. First, we notice that *Postgres* is much more competitive for the queries selecting on the less dependent attribute c4 (Zipf QB.2 and Zipf QB.4), which is explained by the independence assumption. Furthermore, *Postgres* is most competitive for Zipf QB.4, as the assumption of uniformity in the join attribute is less violated by the smaller Zipf parameter. *JoinKDE* clearly outperforms *Sample* for most model sizes in all queries. Interestingly, *AGMS* is very competitive for Zipf QB.1 and Zipf QB.3. We see that the *AGMS* performs better for joins with large join results than small join results, which is consistent to the observations of Rusu and Dobra [43].

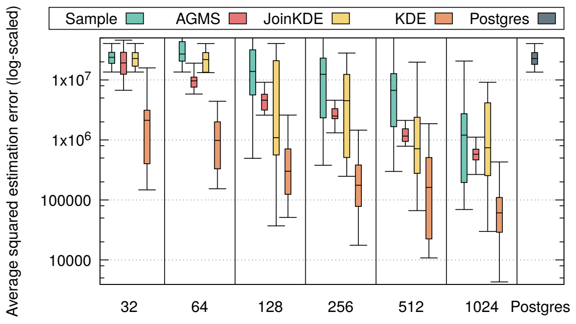
For Zipf QB.3, we notice a significant improvement of the estimation quality for *JoinKDE* and *Sample* for sample size 128 and higher. For Zipf QB.1, we see the same effect happening already at sample size of 64. This shows that these estimators clearly favor relations with skew on the



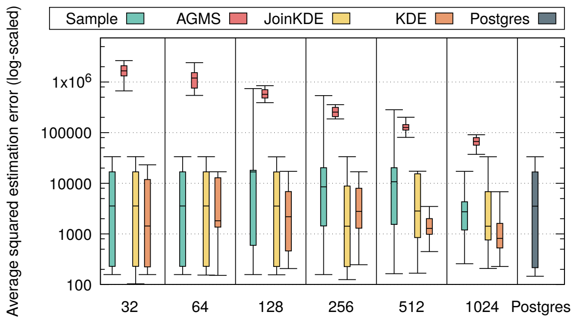
(a) Estimation quality for query Zipf QB.1



(b) Estimation quality for query Zipf QB.2



(c) Estimation quality for query Zipf QB.3



(d) Estimation quality for query Zipf QB.4

Figure 4.5: Estimation quality for query set Zipf QB covering 2-way joins subject to selections

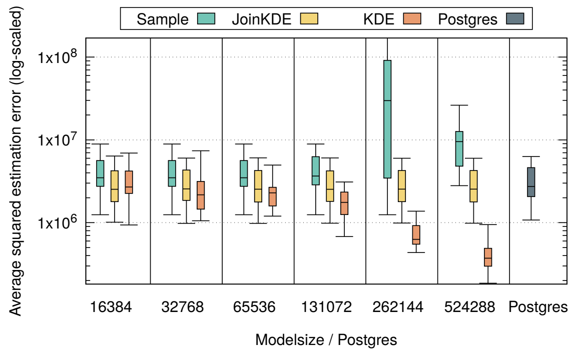
join attribute, as skew increases the chance of finding a join key in both samples.

4.2.2.2 Three-Way Joins

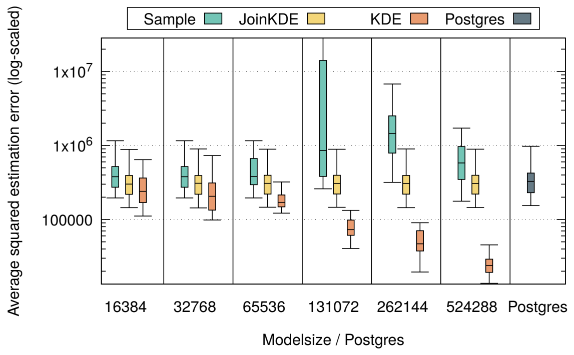
We performed similar experiments for three-way joins to evaluate the performance of our estimator for an increasing number of joins.

We evaluated the estimators for two query sets on the IMDb dataset. We decided to exclude *AGMS* from the evaluation on this dataset, as the sketch computation is unbearably slow for the compared model sizes and the large IMDb relations. Furthermore, we decided to stop the evaluation of model sizes at $2^{19} = 524288$ tuples, which is four times the number of tuples we evaluated for two-way joins. The size corresponds to roughly 10% of the tuples in tables *movie_-keyword*, *title* and *person_info*. It exceeds the number of tuples in *company_name* and corresponds to 1% of tuples in *cast_info*.

Query set **IMDb QD** contains three-way join queries where every table contains exactly one join attribute - so the *JoinKDE* estimator does not have to make an independence assumptions.



(a) Estimation quality for query IMDb QD.1



(b) Estimation quality for query IMDb QD.2

Figure 4.6: Estimation quality for query set IMDb QD covering 3-way joins subject to selections

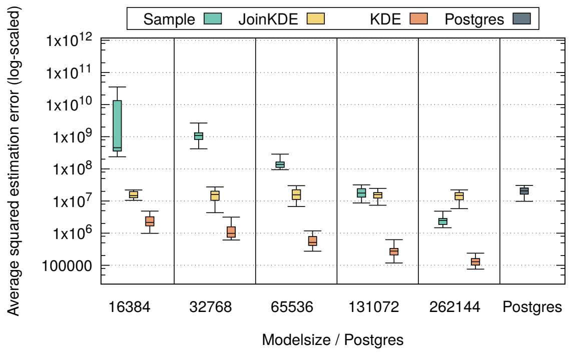
IMDb QD.1: All cast members fulfilling a certain role in an opus from a given year that was associated with a given keyword.

```
select * from title,movie_keyword,cast_info where
              title.id = movie_keyword.movie_id
              and title.id = cast_info.movie_id
              and title.production_year = $x
              and movie_keyword.keyword_id = $y
              and cast_info.role_id = $z
```

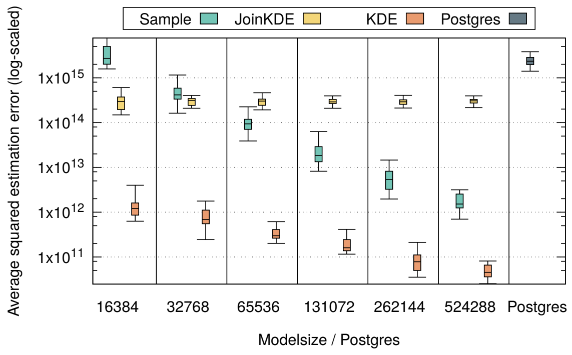
IMDb QD.2: All movie companies of a given type that have produced titles in a given year that were associated with a given keyword.

```
select * from title,movie_keyword,movie_companies where
              title.id = movie_companies.movie_id
              and title.id = cast_info.movie_id
              and title.production_year = $x
              and movie_keyword.keyword_id = $y
              and movie_companies.company_type_id = $z
```

The results are given in Figure 4.6. We find *KDE* being the only estimator that can clearly outperform the estimates provided by *Postgres* given large enough samples. *Postgres* and *KDE* barely differ and provide estimates close to zero as their estimates do not differ significantly from *Sample* for small model sizes. For both queries, the performance of *Sample* deteriorates significantly with an increasing sample size - an effect that we already observed in our experiments using query set IMDb QB. Due to error-driven bandwidth selection *JoinKDE* can avoid the effect but fails to provide better estimates than *Postgres* - even though the maximum evaluated sample size is much larger than the model size required for two-way joins to see an improvement. We observe the effect we predicted in the discussion of our estimator in Section 3.6: As we join over attributes that include a lot of different values and are not significantly



(a) Estimation quality for query IMDb QE.1



(b) Estimation quality for query IMDb QE.2

Figure 4.7: Estimation quality for query set IMDb QE covering 3-way joins subject to selections

skewed, *Sample* and *JoinKDE* both require very large samples to provide good performance for three-way joins. However, *JoinKDE* is capable of falling back to estimates similar to *Postgres* due to error-driven bandwidth optimization.

Query set **IMDb QE** includes three-way join queries where an intermediate table has two join attributes. Thus, the *JoinKDE* estimator has to make an independence assumption to efficiently compute an estimate as described in Section 3.5.

IMDb QE.1: Select companies of a given type that are located in given country and have been associated with an opus from a given year.

```
select * from company_name,title,movie_companies where
              company_name.id = movie_companies.company_id
              and title.id = movie_companies.movie_id
              and company_name.country_code_id = $x
              and title.production_year = $y
              and movie_companies.company_type_id = $z
```

IMDb QE.2: Select a given type of meta information for all cast member with a given role that have worked with companies from a given country.

```
select * from movie_companies,person_info,cast_info where
              movie_companies.movie_id = cast_info.movie_id
              and person_info.person_id = cast_info.person_id
              and movie_companies.company_type_id = $x
              and person_info.info_type_id = $y
              and cast_info.role_id = $z
```

Figure 4.7 gives the result for this query set. We stopped the evaluation on IMDb QE.1 at a sample size of 262144, as the next power of two would have exceeded the size of the smallest involved relation. *KDE* provides superior estimation results for all evaluated sample sizes. Like in the previous experiment, *KDE* and *Postgres* provide similar estimation results. Different to all previous experiments, we find *Sample* outperforming *JoinKDE*, which is due to the introduced independence assumption for the intermediate join table.

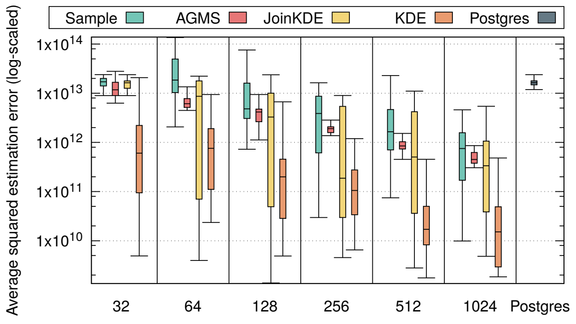
We extended our experiments on the Zipf dataset to three-way joins in query set **Zipf QC**. We applied an additional join to the queries in Zipf QB by joining with a likewise constructed table while maintaining the join and selection attribute used in the query. Figure 4.8 shows the results. Considering Zipf QC.1 and Zipf QC.3, we find that even though another join was introduced, *JoinKDE* and *Sample* do still provide significantly better estimates than *Postgres*. Unlike the previous experiments for three-way joins, all tables are distributed very similarly. Join keys that occurred frequently in the tables, occur even more frequently in the join result as their frequency in the join result is given by the product of the individual frequencies. Thus, these values make the majority of tuples in the join result and queried more frequently. As even small samples are likely to include the most common values. Thus, *Sample* and *KDE* are able to provide much better results compared to the IMDb query sets. Again, we find *AGMS* being very competitive for these two queries, as the join results are large.

Zipf QC.2 and Zipf QC.4 show a different picture. For these queries *AGMS* provides inferior estimates, while all other estimators perform similarly. As the selection attribute c4 does barely dependent on the query attribute, there are many more possible combinations on the selection attributes, which causes problems to all estimators using a sample.

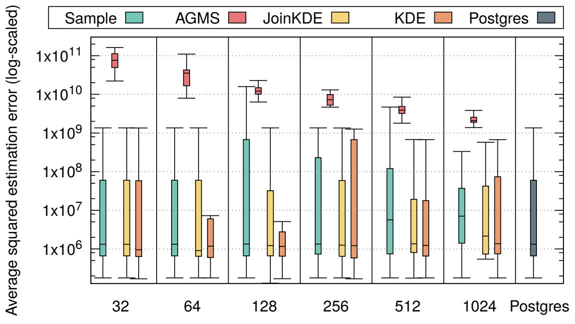
While highlighting an interesting corner case of the estimator, Zipf QD is not representative for a workload that occurs in real-world database system - much more are the queries on the IMDb dataset. We therefore decided not to include further experiments using more than three joins as the experiments on the IMDb dataset show that the required sample sizes would be unreasonable.

4.2.3 Interference of Selections and Join Queries in Bandwidth Optimization

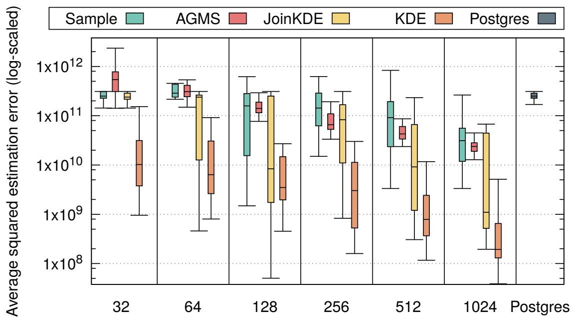
In Section 3.6, we suggested to maintain different bandwidth parameters for joins and selections, as the assumption of equal value sets on join attributes may change the bandwidth to values that are suboptimal for selections. To show this effect, we extracted the samples of size 131072 and the bandwidth parameters used for the evaluation of IMDb QC.1. We selected this query as it clearly violates the assumption of *JoinKDE*. Afterwards, we used the samples and bandwidth parameters for selections on the *movie_keyword* table that cover the join attribute and selection attribute. Figure 4.9 compares the estimation error for bandwidth values optimized on the query workload to the bandwidth values optimized for IMDb QC.1. We see that the provided estimates for the former are close to perfect, while the latter introduces squared



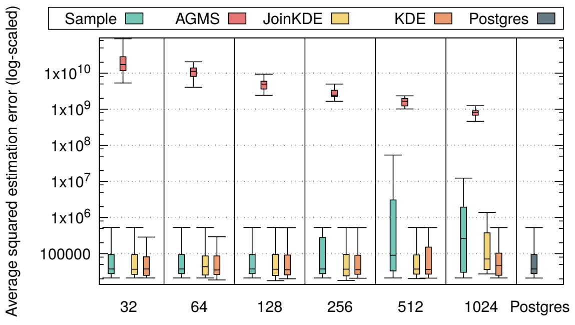
(a) Estimation quality for query Zipf QC.1



(b) Estimation quality for query Zipf QC.2



(c) Estimation quality for query Zipf QC.3



(d) Estimation quality for query Zipf QC.4

Figure 4.8: Estimation quality for query set Zipf QC covering 3-way joins subject to selections

estimation errors worse by two orders of magnitude.

4.3 Execution Time

In this Section, we evaluate our estimator with respect to the execution time. We measured the execution time of the *JoinKDE* and *KDE* estimator for IMDb QC.1 and varying model sizes. The estimators were implemented as sequential C program. The execution environment was a consumer system running Debian Stretch (testing) on an Intel Core i7-3820 processor and 16 GB of main memory. Due to the limited editing time for this thesis, a parallel implementation on a GPU was out of scope. The programs read the required samples from disk and measure the execution time for computing the estimate and the error-independent gradient factor for 100 queries. We performed 20 iterations of these measurements for every evaluated model size and report the average execution time per query.

Figure 4.10 shows the measured execution times for model sizes increasing in powers of two. Note that both axes are logarithmically scaled - the runtime axis with the logarithm of ten and the size axis with the logarithm of two. We added a linear function as a visual aid. The plot confirms that the execution time of both estimators increases linearly in the model size. The

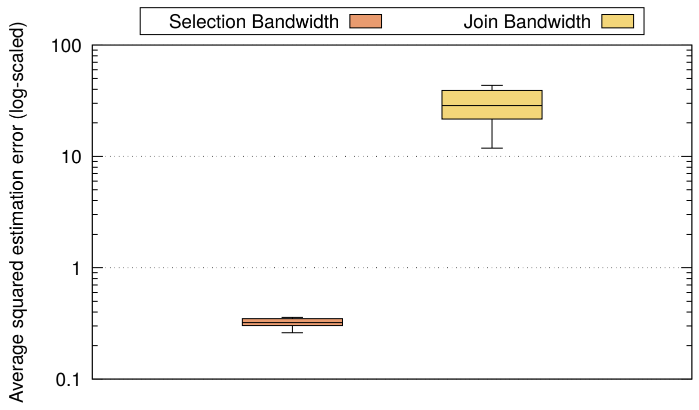


Figure 4.9: Estimation quality of the *KDE* estimator for selections on *movie_keyword* for a bandwidth parameter optimized on the evaluation workload and IMDb QC.1

execution time of *JoinKDE* is roughly three times higher than *KDE*, which is explained by its more involved computations and the two required passes over both samples.

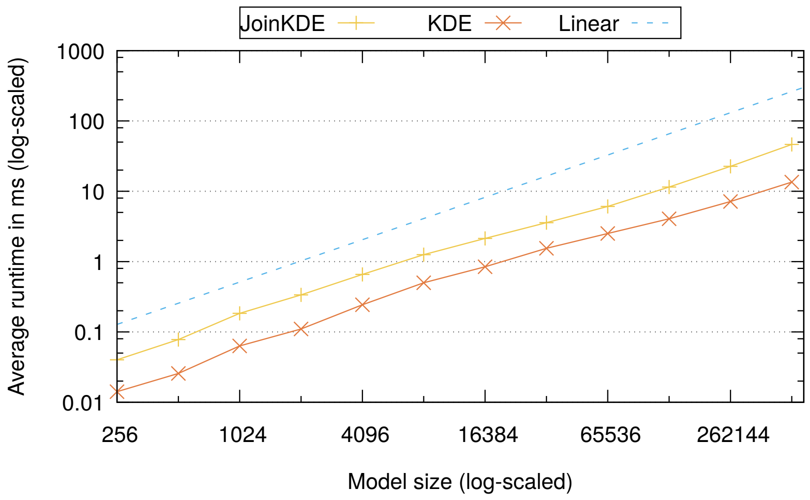


Figure 4.10: Execution time of the JoinKDE and KDE estimator for varying model sizes on IMDb QC.1

5 Conclusion

In this thesis we applied Kernel Density Estimators to the problem of join selectivity estimation. Our approach is based on discrete Kernel Density Estimators, which can provide estimates for the joint frequency distribution of discrete attributes in base tables. The models are constructed from a random sample and the number of distinct values per attribute.

We demonstrated how these models can be used to estimate base table selectivities for selections with conjunctive equality predicates and provided the gradient required to perform error-driven hyperparameter optimization. Our evaluation confirms that a well-trained Kernel Density Estimator model can provide significantly better estimates than naive query evaluation on samples and does at worst provide the same results, as sample evaluation is part of the hyperparameter space of the estimator. Furthermore, Kernel Density Estimator models provided good results in cases the assumptions made in database products like PostgreSQL introduce large errors.

We demonstrated that these models can be applied to for selectivity estimation of two-way equi-joins subject to selections with conjunctive equality predicates. We provided algorithms for computing these selectivities and their error gradients based on two independent Kernel Density Estimator models of the involved based tables. Moreover, we explained how the algorithm can be adapted to multiple joins and sketched a parallel implementation targeting GPUs.

For two-way joins our experimental evaluation shows that well-trained Kernel Density Estimator models are superior to query evaluation on samples as well, as the relationship between the estimators is preserved in our algorithm. Moreover, they provided better results than the AGMS Sketch for most experiments. Compared to the estimates provided by PostgreSQL, we found our estimator providing superior estimates, given the sample size is chosen large enough, and otherwise providing comparable estimates. The required sample size depends on properties of the underlying dataset. It was shown on an artificial dataset that our estimator favors skew inside the join attribute, while requiring larger samples on a real-world dataset, where join keys were not significantly skewed or consisted of unique keys.

For three-way joins the benefits of our approach are less clear. The algorithm delivers admirable estimation quality for experiments on the skewed artificial dataset, but was not able to clearly outperform the simple estimates provided by PostgreSQL for reasonable sample sizes on a real-world dataset. This shows that the estimator scales poorly with the number of joins in realistic scenarios. As making the independence assumptions is required for the general case of more than one join, our estimator could even be outperformed by sample evaluation, which confirms that its usage for more than one join is limited.

In our evaluation, we saw that a Kernel Density Estimator model constructed from the join result provided clearly superior estimates for almost all experiments. While these models are more expensive to construct and hard to maintain under updates, they can be a solution for joins where base table models fail to provide sufficient accuracy. Furthermore, it could be worth exploring if targeting Kernel Density Estimators directly to a specific join by using correlated [51] or end-biased samples [11] can reduce the required sample size.

There is one further aspect we did not cover in our evaluation but deserves further investigation: Discrete Kernel Density Estimators models are easily extended to mixed Kernel Density Estimator models that combine the categorical and continuous kernel to provide estimates for queries including range-selections on real-valued attributes as well as equality selections on discrete attributes. Exploring the possibilities of these models could be an interesting future research topic.

List of Tables

1.1 Notation 6

List of Figures

2.1 A Kernel Density Estimator approximates the underlying distribution of a given dataset (a) by picking a random sample (b), centering local probability distributions (kernels) around them (c), and averaging the local distributions (d). [21] . . 17

2.2 Choosing the bandwidth has a crucial impact on estimation quality. If the bandwidth is too small (a), the estimator over-fits the sample. If it is too large (b), all local information is lost. [21] 18

3.1 The categorical kernel on value 2 with $\lambda = 0.7$ and $C = \{0, 1, 2, 3, 4\}$ 24

3.2 Effect of the bandwidth parameter λ on a Kernel Density Estimator obtained by sampling from a binomial distribution with $n = 4$ and $p = 0.5$. 25

3.3 Merging step in the estimation algorithm 33

3.4 Computation of invariant contributions r and Ω in the GPU algorithm for a toy example 41

3.5 State of the GPU algorithm after performing the binary search step for a toy example 41

3.6 Computation of predicates and prefix sums in the GPU algorithm for a toy example 42

3.7 Computation of $c(v)$ in the GPU algorithm for a toy example 43

4.1 Estimation quality for query set IMDb QA covering selections on base tables . . 52

4.2 Estimation quality for query set IMDb QB covering selections on base tables . . . 53

4.3 Estimation quality for query set Zipf QA covering selections on base tables . . . 54

4.4 Estimation quality for query set IMDb QC covering 2-way joins subject to selections 56

4.5 Estimation quality for query set Zipf QB covering 2-way joins subject to selections 58

4.6 Estimation quality for query set IMDb QD covering 3-way joins subject to selections 59

4.7 Estimation quality for query set IMDb QE covering 3-way joins subject to selections 60

4.8 Estimation quality for query set Zipf QC covering 3-way joins subject to selections 62

4.9 Estimation quality of the *KDE* estimator for selections on *movie_keyword* for a bandwidth parameter optimized on the evaluation workload and IMDb QC.1 . . 63
4.10 Execution time of the JoinKDE and KDE estimator for varying model sizes on IMDb QC.1 63

Listings

3.1 Pseudocode for Step 1 to 3 of the join selectivity estimation algorithm 34
3.2 Pseudocode for Step 4 and 5 of the join selectivity estimation algorithm 35
3.3 Pseudocode for Step 1 to 3 of the gradient algorithm for join selectivities 38
3.4 Pseudocode for Step 4 and 5 of the gradient algorithm for join selectivities 39

Bibliography

Printed References

[1] Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments". In: *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Philadelphia, Pennsylvania, USA: ACM, 1996, pp. 20–29. ISBN: 0-89791-785-5. DOI: 10.1145/237814.237823. URL: http://doi.acm.org/10.1145/237814.237823.

[2] Noga Alon et al. "Tracking Join and Self-join Sizes in Limited Storage". In: *Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS '99. Philadelphia, Pennsylvania, USA: ACM, 1999, pp. 10–20. ISBN: 1-58113-062-7. DOI: 10.1145/303976.303978. URL: http://doi.acm.org/10.1145/303976.303978.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.

[4] Bjorn Blohsfeld, Dieter Korus, and Bernhard Seeger. "A Comparison of Selectivity Estimators for Range Queries on Metric Attributes". In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. SIGMOD '99. Philadelphia, Pennsylvania, USA: ACM, 1999, pp. 239–250. ISBN: 1-58113-084-8. DOI: 10.1145/304182.304203. URL: http://doi.acm.org/10.1145/304182.304203.

[5] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. "STHoles: a multidimensional workload-aware histogram". In: *ACM SIGMOD Record*. Vol. 30. 2. ACM. 2001, pp. 211–222.

[6] Richard H. Byrd et al. "A Limited Memory Algorithm for Bound Constrained Optimization". In: *SIAM Journal on Scientific Computing* 16.5 (1995), pp. 1190–1208. DOI: 10.1137/0916069. eprint: http://dx.doi.org/10.1137/0916069. URL: http://dx.doi.org/10.1137/0916069.

[7] Kaushik Chakrabarti et al. "Approximate Query Processing Using Wavelets". In: *Proceedings of the 26th International Conference on Very Large Data Bases*. VLDB '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 111–122. ISBN: 1-55860-715-3. URL: `http://dl.acm.org/citation.cfm?id=645926.671851`.

[8] Graham Cormode and S. Muthukrishnan. "An Improved Data Stream Summary: The Count-min Sketch and Its Applications". In: *J. Algorithms* 55.1 (Apr. 2005), pp. 58–75. ISSN: 0196-6774. DOI: `10.1016/j.jalgor.2003.12.001`. URL: `http://dx.doi.org/10.1016/j.jalgor.2003.12.001`.

[9] Graham Cormode et al. "Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches". In: *Foundations and Trends® in Databases* 4.1–3 (2011), pp. 1–294. ISSN: 1931-7883. DOI: `10.1561/1900000004`. URL: `http://dx.doi.org/10.1561/1900000004`.

[10] Alin Dobra et al. "Processing Complex Aggregate Queries over Data Streams". In: *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*. SIGMOD '02. Madison, Wisconsin: ACM, 2002, pp. 61–72. ISBN: 1-58113-497-5. DOI: `10.1145/564691.564699`. URL: `http://doi.acm.org/10.1145/564691.564699`.

[11] C. Estan and J. F. Naughton. "End-biased Samples for Join Cardinality Estimation". In: *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*. Apr. 2006, pp. 20–20. DOI: `10.1109/ICDE.2006.61`.

[12] Sumit Ganguly, Minos Garofalakis, and Rajeev Rastogi. "Advances in Database Technology - EDBT 2004: 9th International Conference on Extending Database Technology, Heraklion, Crete, Greece, March 14-18, 2004". In: ed. by Elisa Bertino et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. Chap. Processing Data-Stream Join Aggregates Using Skimmed Sketches, pp. 569–586. ISBN: 978-3-540-24741-8. DOI: `10.1007/978-3-540-24741-8_33`. URL: `http://dx.doi.org/10.1007/978-3-540-24741-8_33`.

[13] Sumit Ganguly et al. "Bifocal Sampling for Skew-resistant Join Size Estimation". In: *SIGMOD Rec.* 25.2 (June 1996), pp. 271–281. ISSN: 0163-5808. DOI: `10.1145/235968.233340`. URL: `http://doi.acm.org/10.1145/235968.233340`.

[14] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database systems - the complete book (2. ed.)* Pearson Education, 2009, pp. I–XXVI, 1–1203. ISBN: 978-0-13-187325-4.

[15] Lise Getoor, Benjamin Taskar, and Daphne Koller. "Selectivity Estimation Using Probabilistic Models". In: *SIGMOD Rec.* 30.2 (May 2001), pp. 461–472. ISSN: 0163-5808. DOI: `10.1145/376284.375727`. URL: `http://doi.acm.org/10.1145/376284.375727`.

[16] Phillip B Gibbons, Yossi Matias, and Viswanath Poosala. *Maintaining a random sample of a relation in a database in the presence of updates to the relation*. US Patent 6,012,064. Jan. 2000.

[17] Dimitrios Gunopulos et al. "Selectivity estimators for multidimensional range queries over real attributes". In: *The VLDB Journal—The International Journal on Very Large Data Bases* 14.2 (2005), pp. 137–154.

[18] Peter J. Haas et al. "Fixed-precision Estimation of Join Selectivity". In: *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. PODS '93. Washington, D.C., USA: ACM, 1993, pp. 190–201. ISBN: 0-89791-593-3. DOI: 10.1145/153850.153875. URL: http://doi.acm.org/10.1145/153850.153875.

[19] Bingsheng He et al. "Relational Joins on Graphics Processors". In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. Vancouver, Canada: ACM, 2008, pp. 511–524. ISBN: 978-1-60558-102-6. DOI: 10.1145/1376616.1376670. URL: http://doi.acm.org/10.1145/1376616.1376670.

[20] Max Heimel, Martin Kiefer, and Volker Markl. "Demonstrating Transfer-Efficient Sample Maintenance on Graphics Cards". In: *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.* 2015, pp. 513–516. DOI: 10.5441/002/edbt.2015.46. URL: http://dx.doi.org/10.5441/002/edbt.2015.46.

[21] Max Heimel, Martin Kiefer, and Volker Markl. "Self-Tuning, GPU-Accelerated Kernel Density Models for Multidimensional Selectivity Estimation". In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015.* 2015, pp. 1477–1492. DOI: 10.1145/2723372.2749438. URL: http://doi.acm.org/10.1145/2723372.2749438.

[22] Ihab F. Ilyas et al. "CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies". In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. SIGMOD '04. Paris, France: ACM, 2004, pp. 647–658. ISBN: 1-58113-859-8. DOI: 10.1145/1007568.1007641. URL: http://doi.acm.org/10.1145/1007568.1007641.

[25] Yannis E. Ioannidis and Stavros Christodoulakis. "On the Propagation of Errors in the Size of Join Results". In: *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*. SIGMOD '91. Denver, Colorado, USA: ACM, 1991, pp. 268–277. ISBN: 0-89791-425-2. DOI: 10.1145/115790.115835. URL: http://doi.acm.org/10.1145/115790.115835.

[27] M Chris Jones, James S Marron, and Simon J Sheather. "A brief survey of bandwidth selection for density estimation". In: *Journal of the American Statistical Association* 91.433 (1996), pp. 401–407.

[28] Martin Kiefer, Max Heimel, and Volker Markl. "Demonstrating Transfer-Efficient Sample Maintenance on Graphics Cards". In: *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.* 2015, pp. 513–516. DOI: `10.5441/002/edbt.2015.46`. URL: `http://dx.doi.org/10.5441/002/edbt.2015.46`.

[29] Per-Ake Larson et al. "Cardinality estimation using sample views with quality assurance". In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM. 2007, pp. 175–186.

[30] Viktor Leis et al. "How Good Are Query Optimizers, Really?" In: *Proc. VLDB Endow.* 9.3 (Nov. 2015), pp. 204–215. ISSN: 2150-8097. DOI: `10.14778/2850583.2850594`. URL: `http://dx.doi.org/10.14778/2850583.2850594`.

[31] Qi Li and Jeff Racine. "Nonparametric estimation of distributions with categorical and continuous data". In: *journal of multivariate analysis* 86.2 (2003), pp. 266–292.

[32] Richard J. Lipton and Jeffrey F. Naughton. "Query Size Estimation by Adaptive Sampling". In: *Selected Papers of the 9th Annual ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. Nashville, Tennessee, USA: Academic Press, Inc., 1995, pp. 18–25. URL: `http://dl.acm.org/citation.cfm?id=211040.211053`.

[33] Richard J. Lipton, Jeffrey F. Naughton, and Donovan A. Schneider. "Practical Selectivity Estimation Through Adaptive Sampling". In: *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*. SIGMOD '90. Atlantic City, New Jersey, USA: ACM, 1990, pp. 1–11. ISBN: 0-89791-365-5. DOI: `10.1145/93597.93611`. URL: `http://doi.acm.org/10.1145/93597.93611`.

[34] Richard J Lipton, Jeffrey F Naughton, and Donovan A Schneider. *Practical selectivity estimation through adaptive sampling*. Vol. 19. 2. ACM, 1990.

[36] Frank Olken. "Random Sampling from Databases". PhD thesis. University of California at Berkeley, 1993.

[37] Matt Pharr and Randima Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005. ISBN: 0321335597.

[40] Naveen Reddy and Jayant R. Haritsa. "Analyzing Plan Diagrams of Database Query Optimizers". In: *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005.* 2005, pp. 1228–1240. URL: `http://www.vldb2005.org/program/paper/fri/p1228-reddy.pdf`.

[41] Florin I Rusu. "Sketches for aggregate estimations over data streams". PhD thesis. University of Florida, 2009.

[42] Florin Rusu and Alin Dobra. "Pseudo-random Number Generation for Sketch-based Estimations". In: *ACM Trans. Database Syst.* 32.2 (June 2007). ISSN: 0362-5915. DOI: 10.1145/1242524.1242528. URL: http://doi.acm.org/10.1145/1242524.1242528.

[43] Florin Rusu and Alin Dobra. "Sketches for Size of Join Estimation". In: *ACM Trans. Database Syst.* 33.3 (Sept. 2008), 15:1–15:46. ISSN: 0362-5915. DOI: 10.1145/1386118.1386121. URL: http://doi.acm.org/10.1145/1386118.1386121.

[44] David W. Scott. *Multivariate Density Estimation - Theory, Practice, and Visualization*. 2. Ed. New York: John Wiley & Sons, 2015. ISBN: 978-1-118-57553-6.

[45] P. Griffiths Selinger et al. "Access Path Selection in a Relational Database Management System". In: *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*. SIGMOD '79. Boston, Massachusetts: ACM, 1979, pp. 23–34. ISBN: 0-89791-001-X. DOI: 10.1145/582095.582099. URL: http://doi.acm.org/10.1145/582095.582099.

[46] Eric J. Stollnitz, Tony D. Derose, and David H. Salesin. "Wavelets for Computer Graphics: A Primer - Part 1". In: *IEEE Computer Graphics and Applications* 15 (1995), pp. 76–84.

[47] Krister Svanberg. "A class of globally convergent optimization methods based on conservative convex separable approximations". In: *SIAM Journal on Optimization* (), pp. 555–573.

[48] Mikkel Thorup and Yin Zhang. "Tabulation Based 4-universal Hashing with Applications to Second Moment Estimation". In: *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '04. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2004, pp. 615–624. ISBN: 0-89871-558-X. URL: http://dl.acm.org/citation.cfm?id=982792.982884.

[49] T Tieleman and G Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural Networks for Machine Learning* 4 (2012).

[50] Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. "Lightweight graphical models for selectivity estimation without independence assumptions". In: *PVLDB* (2011), p. 2011.

[51] David Vengerov et al. "Join Size Estimation Subject to Filter Conditions". In: *Proc. VLDB Endow.* 8.12 (Aug. 2015), pp. 1530–1541. ISSN: 2150-8097. DOI: 10.14778/2824032.2824051. URL: http://dx.doi.org/10.14778/2824032.2824051.

[52] Jeffrey S. Vitter. "Random Sampling with a Reservoir". In: *ACM Trans. Math. Softw.* 11.1 (Mar. 1985), pp. 37–57. ISSN: 0098-3500. DOI: 10.1145/3147.3165. URL: http://doi.acm.org/10.1145/3147.3165.

[53] Feng Yu et al. “CS2: a new database synopsis for query estimation”. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*. Ed. by Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias. ACM, 2013, pp. 469–480. ISBN: 978-1-4503-2037-5. URL: `http://dl.acm.org/citation.cfm?id=2463676`.

Online References

[23] *IMDb: Alternative Interfaces*. URL: `http://www.imdb.com/interfaces` (visited on 03/23/2016).

[24] *IMDbPY*. URL: `http://imdbpy.sourceforge.net/` (visited on 03/23/2016).

[26] Steven G. Johnson. *The NLopt nonlinear-optimization package*. URL: `http://ab-initio.mit.edu/nlopt` (visited on 03/23/2016).

[35] *NVIDIA - Parallel Thread Execution ISA Version 4.3*. URL: `http://docs.nvidia.com/cuda/parallel-thread-execution` (visited on 03/23/2016).

[38] *PostgreSQL: Documentation: 9.5: How the Planner Uses Statistics*. URL: `www.postgresql.org/docs/9.5/static/planner-stats-details.html` (visited on 03/23/2016).

[39] *PostgreSQL: The world's most advanced open source database*. URL: `http://www.postgresql.org/` (visited on 03/23/2016).