Selected Problems in Cardinality Estimation

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von
Magnus Müller, M.Sc.
aus Tübingen

Mannheim, 2022

Dekan:	Dr. Bernd Lübcke, Universität Mannheim
Referent:	Prof. Dr. Guido Moerkotte, Universität Mannheim
Korreferent:	Prof. Dr. Michael Grossniklaus, Universität Konstanz

Tag der mündlichen Prüfung: 21.12.2022

Abstract

Cardinality estimation remains a critical task in query processing. Query optimizers rely on the accuracy of cardinality estimates when generating execution plans, and, in approximate query answering, estimated cardinalities affect the quality of query results.

In this thesis, we present multiple new cardinality estimation techniques. The techniques differ vastly by the query under consideration. For single relation queries, we use the principle of maximum entropy to combine information extracted from samples and histograms. For join size estimation, we rely on a model that requires one to find estimates for the intersection size of join attributes. For queries with multiple joins, sketches serve as compact representations of join results that are combined via a data structure that approximates the joint frequency distribution of join attributes. In addition, we present a technique to transform selection predicates into a representation that allows estimators based on machine learning to effectively learn query result cardinalities.

For each cardinality estimator presented in this thesis, we precisely define its problem scope, the construction process, and how to obtain estimates. Then, we compare to state-of-the-art cardinality estimators and run a thorough evaluation with queries over multiple data sets. Based on our observations, we analyze the strengths and limitations of each of our cardinality estimators and identify its preferred use case.

Zusammenfassung

Kardinalitätsabschätzung ist weiterhin ein gewichtiges Thema im Bereich Anfragebearbeitung. Für die Generierung von Auswertungsplänen verlassen sich Anfrageoptimierer auf die Güte von Kardinalitätsschätzern. Außerdem beeinflussen Kardinalitätsschätzungen die Qualität von Anfrageergebnissen im Bereich approximative Anfragebearbeitung.

Diese Dissertation präsentiert mehrere neue Techniken zur Kardinalitätsabschätzung. Diese Techniken unterscheiden sich stark in den Anfragen, auf die sie sich beziehen. Für einzelne Relationen greifen wir auf die Maximum-Entropie-Methode zurück, um Informationen aus Stichproben und Histogrammen zu kombinieren. Zur Schätzung von Joinergebnisgrößen verwenden wir ein Modell, welches auf Schätzungen von Attributschnittmengen aufbaut. Für Anfragen mit mehreren Join Operationen dienen uns Sketch-Datenstrukturen zur kompakten Abbildung von Joinergebnissen. Über eine weitere Datenstruktur, welche die gemeinsame Häufigkeitsverteilung von Joinattributen approximiert, können die Sketch-Datenstrukturen zu einer Kardinalitätsschätzung kombiniert werden. Über dies hinaus präsentieren wir eine Technik zur Transformation von Selektionsprädikaten in eine Form, welche es Kardinalitätsschätzern, die maschinelles Lernen anwenden, erlaubt, Ergebniskardinalitäten erfolgreich zu lernen.

Für jeden der in dieser Dissertation vorgestellten Kardinalitätsschätzern definieren wir seinen Anwendungsbereich, die notwendigen Schritte zur Erstellung sowie das Verfahren, um eine Schätzung zu erhalten. Anschließend ziehen wir einen Vergleich zu anerkannten existierenden Kardinalitätsschätzern und unterziehen die Kardinalitätsschätzer einer gründlichen Evaluierung an Hand von Anfragen über verschiedene Datensätze. Basierend auf den Beobachtungen, die wir dabei erheben, analysieren wir die Stärken und Schwächen jedes unserer Kardinalitätsschätzer und identifizieren seinen jeweiligen empfohlenen Anwendungsbereich.

Contents

1 Introduction **11**

2 Query Processing **13**

3 Prerequisites **17**
3.1 Cardinality and Selectivity 17
3.2 Cardinality Estimation 19
3.2.1 Error Measures 19
3.2.2 Estimators 22

4 State of the Art **25**
4.1 Sampling 25
4.2 Histograms 27
4.3 Sketches 27
4.4 Machine Learning 29
4.5 Shortcomings 30

5 Selection Predicates over Single Relations **33**
5.1 Introduction 33
5.2 Preliminaries 34
5.2.1 Predicates and Selectivities 34
5.2.2 Relation between β- and γ-selectivities 35
5.2.3 Matrix Representation 36
5.2.4 A Linear System Induced by Synopses 37
5.3 Existing Approaches 38
5.4 Combined selectivity estimation 40
5.4.1 Sampling Bounds 40
5.4.2 Synopses Bounds 44
5.4.3 Estimating Selectivities: The Optimization Problem . . . 45
5.4.4 Solving the Optimization Problem 47
5.5 Experimental Evaluation 52
5.5.1 Accuracy 53
5.5.2 Dimensionality of the Given Synopses and Sample Size . . 67
5.5.3 Information Gain 68

5.5.4 Confidence Level of Sampling Bounds 69
5.5.5 Runtime 76
5.6 Summary 77

6 Sketches for Intersection Size Estimation 79
6.1 AKMV Sketch 79
6.2 HyperLogLog 81
6.2.1 HyperLogLog Sketch 81
6.2.2 HyperLogLog Intersection Size Estimation 82
6.3 Evaluation 84
6.3.1 Hash Function 84
6.3.2 Intersection Size Estimation 85

7 Two-way Joins 97
7.1 Introduction 97
7.2 Join Size Estimation with Bucket Sketches 100
7.2.1 Bucket Sketch 100
7.2.2 Average Multiplicity Estimation 105
7.2.3 Error Analysis 107
7.2.4 Extensions and Limitations 109
7.3 Evaluation 109
7.3.1 Data Generation 110
7.3.2 Accuracy Join Size Estimation 111
7.3.3 Memory Consumption 120
7.3.4 Time Consumption 120
7.3.5 Space-Accuracy Tradeoff 121
7.3.6 Equi Width vs. other Partitioning Schemes 122
7.3.7 Intersection Size Estimation Subject to Predicates 124
7.4 Summary 128

8 Multiple Joins 129
8.1 Introduction 129
8.2 New Operations for AKMV Sketches 130
8.3 Translation Grids 133
8.3.1 Structure 134
8.3.2 Construction 134
8.3.3 Translation 135
8.3.4 Estimation Properties of Translation Grids 136
8.4 Evaluation 138
8.5 Summary 144

9 Query Featurization 147
9.1 Introduction 147
9.2 Preliminaries 149
9.2.1 State of the Art in Query Featurization 149
9.2.2 Machine Learning Models 151

9.3 Query Featurization . . . 153
9.3.1 Range Predicate Encoding . . . 155
9.3.2 Universal Conjunction Encoding . . . 155
9.3.3 Limited Disjunction Encoding . . . 159
9.4 Implementation . . . 161
9.4.1 Local Models . . . 161
9.4.2 Global Models . . . 162
9.5 Evaluation . . . 162
9.5.1 Quality under All QFT × Model Combinations . . . 164
9.5.2 Comparison with Other Estimators . . . 167
9.5.3 Feature Vector Length . . . 169
9.5.4 Concept Drift . . . 170
9.5.5 Training Convergence . . . 172
9.5.6 Time & Memory Consumption . . . 173
9.6 Discussion . . . 173
9.7 Summary . . . 174

10 Conclusion and Outlook 177

Chapter 1

Introduction

The problem of estimating cardinalities, i.e., query result sizes, remains a critical task in the area of database systems. Query optimizers use cardinality estimates as crucial parameters to cost functions which determine query plan selection. In particular, the query optimizer requests estimates for the cardinality of intermediate results during plan generation. While the earliest cardinality estimation techniques date back to the advent of cost-based query optimization in 1979 [78], cardinality estimation remains an active research topic [23, 47, 54]. Note that cardinality estimation is not a purely academic problem. Both theoretical [66] and empirical findings [53] indicate that query execution times are reduced when cardinality estimates improve. This claim is further supported by ongoing efforts of the database industry to invent more accurate cardinality estimation techniques [9].

This thesis contributes new cardinality estimation techniques for specific queries or relational algebra expressions. The contribution includes cardinality estimation techniques for queries over different numbers of relations, from a single relation over a join expression to multiple joins. We believe that there are no real-world queries without selection predicates. Hence, all our estimation techniques support selection predicates. The cardinality estimation techniques presented in this thesis differ vastly by the query under consideration. For single relation queries, we use the principle of maximumum entropy to combine information extracted from samples and histograms. For join size estimation, we rely on a model that requires one to find estimates for the intersection size of join attributes. For multiple joins, sketches serve as compact representations of join results that are combined via a data structure that approximates the joint frequency distribution of join attributes. In addition, we transform selection predicates into a representation that allows estimators based on machine learning to effectively learn query result cardinalities.

The thesis is structured as follows: Section 2 outlines how database systems process queries and where cardinality estimates are used. Section 3 presents prerequisites of the following chapters, e.g., discusses how we measure estimation errors. Then, in Section 4, we discuss existing state-of-the-art cardinality

estimation techniques and what their shortcomings are. Sections 5, 7, 8, and 9 form the core of this work and present the cardinality estimation techniques that this thesis contributes. The unmentioned Section 6 introduces and analyses data structures extensively used in Sections 7 and 8. Finally, in Section 10, we draw a conclusion.

Chapter 2

Query Processing

In this section, we present foundations of relational database systems and illustrate how queries are processed. We do so by following the flow of a query from the point in time it enters the database system to the point at which its result is delivered.

All database systems allow users to query the stored data by formulating a query statement using a query language. Most query languages are declarative, in the sense that a query describes a query result but not how to compute it. The most common query language in general, and in particular for relational data, is SQL. An example query written in SQL is:

```
SELECT A, B, C
FROM R1, R2, F
WHERE R1.K = F.R1 AND R2.K = F.R2
AND R.A < 5 AND R.B > 7 AND R2.C = 'X';
```

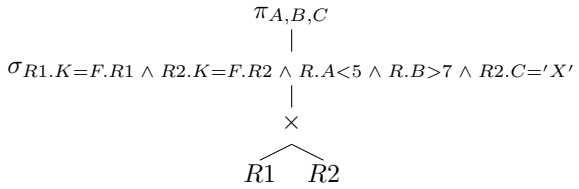
To compute the result, a query must be taken as input and be processed by the database system. On a high level, the database system architecture consists of two components, the compile-time system (CTS) and the run-time system (RTS). The flow of a query through the database system is as follows [62]:

$$\xrightarrow{Query} \textbf{CTS} \xrightarrow[Plan]{Execution} \textbf{RTS} \xrightarrow{Query\ Result}$$

That is, as a query enters the database system, the CTS translates the query to an *execution plan*. An execution plan specifies the steps that must be taken to compute the query result.

As we will see, cardinality estimation, the topic of this thesis, plays a crucial role for the creation of the execution plan. Hence, we take a closer look at the CTS. When the query enters the CTS, it is parsed and translated into an *internal representation* [62, 73]. In textbook query optimization, this process is called *canonical translation* and *operator trees* serve as internal representations. An operator tree is an abstract representation of an execution plan. The result

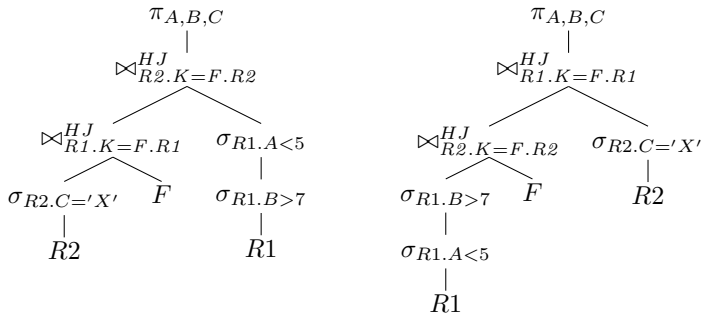
of the canonicial translation of the above example query is the following operator tree:



The initial operator tree leaves space for optimization. The optimization step is performed by the *query optimizer*, which, traditionally, builds on three components [53]:

1. plan enumerator,
2. cost model,
3. cardinality estimator.

The task of the plan enumerator is to enumerate operator trees that correspond to correct execution plans. For instance, for the example query, the plan enumerator may generate the following two operator trees:



In the operator trees, $\bowtie^{HJ}_p$ denotes a hash join with predicate p, where we assume the hash table is built on the right input. Note that the two operator trees differ in both join order and selection predicate order. Which one is preferable? Of course the operator tree that translates to the execution plan we can evaluate faster. Or, in general, the one that has cheaper cost subject to some metric that is reflected in a *cost model*. While the true, measured cost is known only *after* plan execution, the goal of query optimization is to identify the cheapest plan - or at least a cheap plan - before executing any plan. Hence, the cost of plans must be predicted.

Usually, cost models are (deterministic) functions in the number of rows, i.e., the *cardinality*, of their inputs. For instance, a simple cost function is C_{out}.

For an operator tree T, $C_{out}(T)$ is the sum of all output cardinalities. We define C_{out} as

$$C_{out}(T) := \begin{cases} |T|, & \text{if } T = op(T_{inner}), op \in \{\sigma, \pi, \Gamma\} \\ |T| + C_{out}(T_{left}) + C_{out}(T_{right}), & \text{if } T = T_{left} \bowtie T_{right} \\ 0, & \text{if } T \text{ is base relation} \end{cases}$$

Note how the outcome of C_{out} is completely determined by the cardinality of intermediate results. Clearly, since the cardinalities of intermediate results are generally unknown, they must be estimated. Only then the query optimizer can identify the execution plan with the lowest predicted cost. Hence, cardinality estimation - the topic of this thesis - is a critical factor for successful query optimization. We detail on the definition of cardinality and cardinality estimation in the next section.

A side note on cost functions: If you ask a database user about her expectations of a database system, the answer probably involves her queries to have low execution times. Clearly, C_{out} does not directly describe execution times, after all its output is not even a time unit. However, a cost function that assigns the lowest predicted cost to the fastest plan is useful since the query optimizer does exactly what the database user expects. In fact, Wolf et al. [91] observed a high positive correlation between C_{out} and execution time.

Coming back to the big picture: Once the query optimizer has chosen an operator tree, it is translated into the execution plan - recall the difference between the two described above. The execution plan is then forwarded to the RTS. Within the RTS, a component called the *query evaluation engine* evaluates the execution plan and returns the query result. While many techniques to implement the RTS could be discussed, we do not go into depth since this is not necessary in the context of this thesis.

To sum up, this section presents the high-level architecture of database systems and the critical role of cardinality estimation in query processing.

Chapter 3

Prerequisites

In this section, we discuss relevant prerequisites. In particular, we discuss the foundations of *cardinality estimation*. First, Section 3.1 focuses on the *cardinality* aspect. Then, we focus on the *estimation* part.

3.1 Cardinality and Selectivity

In the context of relational database systems, relational algebra expressions represent (sub-)queries, e.g., $\sigma_p(R)$ or $R \bowtie_q S$, where R and S are relations and p and q are predicates. Since relational algebra is closed under all operations, the result of a relational expression is always a relation. Throughout this thesis, we operate on bags or multisets, like in SQL.

Given a database instance, each relational expression E has a corresponding *cardinality*

$$|E|,$$

i.e., the number of (not necessarily distinct) tuples in the result from evaluating E.

The term *selectivity* is closely related to cardinality. In general, a selectivity s is a ratio in $[0, 1]$. Selectivities are defined as relative cardinalities

$$\frac{|E_1|}{|E_2|},$$

where E_1, E_2 are relational expressions with $|E_1| \leq |E_2|$. Selectivities are used in the context of selection predicates and join predicates. The selectivity s_p of selection predicate p applied to relation R (not neccessarily a base relation) is defined as

$$s_p = \frac{|\sigma_p(R)|}{|R|}. \tag{3.1}$$

For a join $R \bowtie_p S$, where R, S are relations and p the join predicate, the join selectivity is defined as

$$s_p = \frac{|R \bowtie_p S|}{|R \times S|} = \frac{|R \bowtie_p S|}{|R| \cdot |S|} \tag{3.2}$$

When $s_p < s_q$, we say that predicate p is more selective than predicate q. Note the perhaps counterintuitive jargon: The more selective a predicate is, the lower its selectivity.

The selectivity s_p of some selection predicate p over some relation R can be interpreted in two ways:

1. The ratio of tuples in R that satisfy p.

2. The probability that an arbitrary tuple from R satisfies p. Hence, p is treated as an *event*, in the sense of probability theory.

Note that, again, R can either be a base relation or the result of an arbitrary relational algebra expression. For join expressions $R_1 \bowtie_p R_2$, the ratio or probability is with respect to $|R_1 \times R_2|$.

Selectivities allow us to express the cardinality of some complex relational algebra expression in terms of simpler expressions. For instance, consider the cardinality of the following expression:

$$|\sigma_{p_1}(R_1) \bowtie_{p_J} \sigma_{p_2}(R_2)|$$

which, according to Equation (3.2), equals to

$$s_{p_J} \cdot |\sigma_{p_1}(R_1)| \cdot |\sigma_{p_2}(R_2)|$$

and, according to Equation (3.1), this equals to

$$s_{p_J} \cdot (s_{p_1}|R_1|) \cdot (s_{p_2}|R_2|).$$

In relational algebra expressions that contain multiple predicates, one predicate might influence the selectivity of another predicate. In the sense of selectivities as probabilities: predicate outcomes are dependent events. The above notation of selectivities is not well-suited to denote the selectivity of one predicate conditioned on another predicate. Hence, we use $s(p_1|p_2)$ to denote that the selectivity of predicate p_1 given predicate p_2 is true. Then, for the expression $|\sigma_{p_1}(\sigma_{p_2}(R))|$ we have:

$$s(p_1 \wedge p_2) \cdot |R| = s(p_2|p_1) \cdot s(p_1) \cdot |R|$$

We will come back to the cardinality of a conjunction of predicates over one relation in Section 5.

Name	Definition	Minimum	Symmetric	Relative
Absolute error	$\|x - e\|$	0		
Squared error	$(x - e)^2$	0		
Relative error	$\|\frac{\|x-e\|}{x}$	0		✓
Q-error	$\max(\frac{x}{e}, \frac{e}{x})$	1	✓	✓
P-error	$\frac{e-x}{\min(x,e)}$	0	✓	✓
Log change	$\log(\frac{e}{x})$	0	✓	✓

Table 3.1: List of common error measures

3.2 Cardinality Estimation

We defined cardinality in the previous section. This section discusses aspects of finding an estimate e for a cardinality x.

While state-of-the-art cardinality estimation techniques are presented in Section 4, this section focuses on criteria for good cardinality estimation and describes the old but versatile System R estimator.

3.2.1 Error Measures

To measure the accuracy of an estimator, e.g., a cardinality estimation technique, a measure of accuracy is needed. We refer to this measure of accuracy as *error measure*. An error measure is a function in a value x and its estimate e that returns an error value. Table 3.1 lists commonly used error measures. In general, an error measure has a unique minimum error, 0 for most error measures. To evaluate the quality of an estimator, one usually either (1) theoretically analyses a function that describes the distribution of errors around true values, or (2) empirically evaluate observed errors. – Statisticians call the second approach sampling from the (perhaps unknown) error distribution function.

In any case, errors are aggregated to evaluate the quality of an estimator. For instance, in classic statistics it is common to analyze the bias and variance of an estimator. However, other perspectives on an estimator can be thought of and are, in many practical applications, more relevant. In particular, quantile errors, i.e., the error observed (either theoretically or empirically) at some quantile. Common quantile errors include the median error, 95% error, 99% error or the maximum/100% error. For most estimators, the latter is of interest in empirical tests only since most error distribution functions are unbounded.

In this work, estimators are evaluated using various techniques. However, the bulk of the analysis relies on empirical evaluations, using real-world data sets, of 95% and 99% quantile errors where the q-error serves as error measure. We justify this choice:

Why empirical over theoretical evaluations? (1) The main problem is that the true error distribution function is usually unknown. Approximations based on the normal distribution or central limit theorem usually lead to distribution

functions that do not describe the reality well enough to justify choosing one cardinality estimator over another. (2) In addition, it cannot be assumed that q-errors are normally distributed.
Why quantiles over averages? (1) Quantiles allow for clear statements: For instance, 99% of the errors lay below the value of the 99th quantile. Averages do not allow for such statements. Unlike quantiles, averages do not help us to understand the distribution of errors and, hence, it remains unclear if the average error represents a typical error. (2) In the corporate world, quantiles are commonly used in service level agreements. For instance, Microsoft Azure's Cosmos DB makes certain latency guarantees for the 99th quantile of read and write operations [1, 6]. To offer such guarantees, knowledge of quantiles is essential.
Why q-error? Why use the q-error $qe(x, e) = \max\left(\frac{e}{x}, \frac{x}{e}\right)$ to measure the deviation of estimate e from value x instead of commonly known errors like the absolute error $|e-x|$ or relative error $|e-x|/x$? This is not a marginal question. In the following, we assume $e, x > 0$ since some error measures are undefined otherwise. Note that this is no restriction in cardinality estimation.

Argument 1: The absolute error and relative error are insufficient error measures in the context of model selection in general [84], not just in the area of query optimization. While it may be obvious to see that an absolute error metric, like $|e - x|$, is a bad choice, it is harder to see that the commonly used relative error $|e - x|/x$ is insufficient for its asymmetry. Choosing $e = 0$, one notes that the worst possible error of an underestimate is 1. However, in case of overestimation the error is unbounded. This property of the relative error causes an unacceptable systematic preference for estimators that underestimate. In model selection, an error metric should be relative and asymmetric [84], as is the case for the q-error.

Argument 2: In the following, we present an argument from [62] for choosing the q-error as the right error measure in the context of cardinality estimation for query optimization.

Assume, without loss of generality, that our goal is to estimate the cardinality of multiple joins over filtered relations:

$$\sigma_{p_1}(R_1) \bowtie_{p_{1,2}} \ldots \bowtie_{p_{n-1,n}} \sigma_{p_n}(R_n)$$

Denote by s_i the selectivity of selection predicate p_i. Further denote by $s_{i,j}$ the join selectivity of join predicate $p_{i,j}$ if it exists and let $s_{i,j} = 1$ otherwise. Let $N = \{1 \ldots n\}$ be the set of indices from the above relational algebra expression. Denote by E an expression that joins an arbitrary subset $S \subseteq N$ of filtered relations. Then, when assuming predicate independence, the cardinality of E is:

$$|E| = (\prod_{i \in S} s_i)(\prod_{i,j \in S} s_{i,j})(\prod_{i \in S} |R_i|)$$

Denote by $\widehat{s_i}$ the estimate for s_i. Then

$$\begin{aligned}|\hat{E}| =&(\prod_{i\in S}\widehat{s_i})(\prod_{i,j\in S}s_{i,j})(\prod_{i\in S}|R_i|)\\ =&(\prod_{i\in S}s_i/s_i)(\prod_{i\in S}\widehat{s_i})(\prod_{i,j\in S}s_{i,j})(\prod_{i\in S}|R_i|)\\ =&(\prod_{i\in S}\widehat{s_i}/s_i)|E|\end{aligned}$$

Observe that some $i \in S$ correspond to overestimates $\widehat{s_i}/s_i > 1$, while some others correspond to underestimates $\widehat{s_i}/s_i < 1$. However, during plan generation using dynamic programming all $S \subseteq N$ are considered. Hence, since for some $S \subseteq N$ only underestimates and for some others only overestimates occur, we cannot expect that errors cancel out. Instead, to minimize the estimation error, we should minimize each miss-estimation factor:

$$\prod_{i\in S}\max\left(\frac{\widehat{s_i}}{s_i},\frac{s_i}{\widehat{s_i}}\right)$$

This product is minimized by minimizing each of its factors. Since the factors are exactly the definition of the q-error, no other error measure is suited better to minimize error propagation of selectivity estimates of selection predicates. Furthermore, since the above reasoning can be repeated for the join predicates, the result is not restricted to selection predicates.

Argument 3: Another argument for the q-error is a theorem that links estimated query plan costs to true query plan costs via the q-error [44]. Recall that the cost of a query plan is a function in the cardinality of its subplans, cf. C_{out} from Section 2. In the following, we present the theorem:

Theorem 1. *Let $\Sigma = \{E_1 \dots E_n\}$ be the query plans for some query $\mathcal{Q}$ from which the query optimizer chooses. Denote by $C(E_i)$ the estimate for the cost $M(E_i)$ of some query plan E_i. If for all $E_i \in \Sigma$,*

$$qe(M(E_i), C(E_i)) \leq k \tag{3.3}$$

for some constant k, then

$$qe(M(E_{chosen}), M(E_{opt})) \leq k^2 \tag{3.4}$$

where

$$\begin{aligned}E_{opt} &:= arg\min_{E_i\in\Sigma} M(E_i)\\ E_{chosen} &:= arg\min_{E_i\in\Sigma} C(E_i)\end{aligned}$$

denote the cheapest plan and the plan chosen by the query optimizer, respectively.

Proof. Without loss of generality, assume E_{opt} and E_{chosen} are unique, i.e., there exists exactly one minimum. Clearly, the query optimizer chooses E_{chosen} over E_{opt}, despite $M(E_{opt}) < M(E_{chosen})$, since $C(E_{chosen})$ is an underestimate or (not exclusive) $C(E_{opt})$ is an overestimate. In any case, by Equation (3.3),

$$\begin{aligned} M(E_{chosen}) &\leq kC(E_{chosen}) \\ M(E_{opt}) &\geq (1/k)C(E_{opt}) \end{aligned}$$

It follows that

$$\begin{aligned} qe(M(E_{opt}), M(E_{chosen})) &= \max\left(\frac{M(E_{opt})}{M(E_{chosen})}, \frac{M(E_{chosen})}{M(E_{opt})}\right) \\ &= \frac{M(E_{chosen})}{M(E_{opt})} \\ &\leq \frac{kC(E_{chosen})}{(1/k)C(E_{opt})} \\ &\leq \frac{kC(E_{opt})}{(1/k)C(E_{opt})} \\ &= k^2 \end{aligned}$$

which proves Equation (3.4). □

3.2.2 Estimators

To give a detailed description of a classic and versatile estimator, this section describes the approach by System R. The section also highlights the many (unrealistic) assumptions upon which the cardinality estimator builds.

The publication of the System R cardinality estimator [78] dates back to 1979 and is still used by some database systems today. In System R, the cardinality of a relational algebra expression

$$\sigma_p(\times_{i=1}^{n} R_i),$$

where R_i is a base relation or the result of a subquery, is estimated as

$$s(p)\prod_{i=1}^{n}|R_i|.$$

The predicate p might be complex, like $A{>}5 \wedge B{=}$'Mannheim' $\wedge (C{=}1 \vee C{=}2) \wedge K{=}FK$. To derive a selectivity estimate for such predicates, Selinger et al. propose the formulas given in Table 3.2. First, note that we can think of types of predicates that are not listed in the table. Hence, the System R cardinality estimator is not complete, on which we detail at the end of this section. Let A and B denote attributes, c, c_1, c_2 denote constants, L denote a list of values, and Q denote a subquery. For some attribute A, $\min_A, \max_A, d_A$ denote A's

Predicate	Estimate	Precondition	Assumption
$A = c$	$1/d_A$ $1/10$	d_A known	uniform distribution
$A > c$	$\frac{\max_A - c}{\max_A - \min_A}$ $1/3$	$\max_A, \min_A$ known	unif. dist. & unif. spread
$c_1 \leq A \leq c_2$	$\frac{c_2 - c_1}{\max_A - \min_A}$ $1/4$	$\max_A, \min_A$ known	unif. dist. & unif. spread
$\neg p$	$1 - s(p)$		
$p_1 \wedge p_2$	$s(p_1) \cdot s(p_2)$		predicate independence
$p_1 \vee p_2$	$s(p_1) + s(p_2) - s(p_1 \wedge p_2)$		inclusion-exclusion principle
$A = B$	$1/\max(d_A, d_B)$ $1/d_X$ $1/10$	d_A, d_B known $d_X, X \in \{A, B\}$ known	key/foreign-key join
A IN L	$\min(0.5,\ s(A = c) \cdot \lvert L \rvert)$		elements in L are distinct
A IN Q	$\lvert Q \rvert\ /\ \prod_{R_i \in Q} \lvert R_i \rvert$		selectivity linear in $\lvert Q \rvert$, referential integrity

Table 3.2: The System R selectivity estimation formulas by Selinger et al. [78].

minimum value, maximum value, and number of distinct values, respectively. For several types of predicates, Table 3.2 lists more than one possible estimate. For instance, the selectivity of $A = c$ is estimated as $1/d_A$ if d_A is known, and as $1/10$ in absence of any knowledge. The estimate with the strictest precondition is always preferable and listed first for each each type of predicate. Note that (1) no precondition requires more knowledge than $\min_A, \max_A, d_A$, for some attribute A. Further note that, (2) even in absence of any knowledge, a cardinality estimate can be given.

The rightmost column in Table 3.2 lists assumptions under which an estimate is accurate. We take a closer look at these assumptions in the following. Note that the selectivity estimate of all selection predicates builds on the uniform distribution assumption, i.e., each $v \in A$ has the same frequency. Also, for each constant c in a selection predicate, to which values of an attribute A are compared to, it is assumed that $c \in [\min(A), \min(A)]$. In addition, for range predicates, uniform spread is assumed, i.e., the values $v \in A$ are spread uniformly over the A's domain $\max(A) - \min(A) + 1$. Multiple selectivities are combined under the predicate independence assumption. In particular, the selectivity of the conjunction $p_1 \wedge p_2$ is estimated as the product of the two unconditional selectivities $s(p_1) \cdot s(p_2)$, rather than $s(p_1) \cdot s(p_2|p_1)$.

For disjunctions, the inclusion-exclusion principle is applied, which is always a valid rewrite. However, note that the inclusion-exclusion principle replaces the selectivity of one disjunction of n predicates by $2^n - 1$ selectivities of conjunctive predicates, i.e., all non-empty subsets of the n predicates. This can be seen by applying the rewrite in Table 3.2 recursively. After the rewrite, the selectivity of $2^n - n - 1$ conjunctive predicates must be estimated, and to each of them the predicate independence assumption is applied. Hence, selectivity

estimation for both conjunctions and disjunctions of predicates relies on the predicate independence assumption.

Note that A `IN` L can be rewritten to $\bigvee_{v_i \in L} A{=}v_i$. However, instead of applying the estimate for disjunctions, assuming that the elements in L are distinct allows for a simpler estimate. If all elements in L are distinct, then no two predicates are true at the same time and, hence, the selectivity of all conjunctions is zero. Then, $s(A \texttt{ IN } L) = s(\bigvee_{v_i \in L} A{=}v_i) \overset{dist.}{=} \sum_{v_i \in L} s(A{=}v_i) \overset{unif.}{=} s(A{=}c) \cdot |L|$.

For a predicate of the form A `IN` Q, System R assumes that the selectivity is linear in $|Q|$. First, note that the estimate $|Q| \,/\, \prod_{R_i \in Q} |R_i|$ is indeed a number in $[0, 1]$ since the cardinality of Q is at most the product of the cardinalities of its relations. For the case when Q is of the form $\pi_B(\sigma_p(S))$ and the referential integrity $\pi^D_A(R) \subseteq \pi^D_B(S)$ holds, Selinger et al. justify the estimate, which coincides with $s(p)$. If $s(p) = 1$, then also the selectivity estimate is 1. This is accurate since $R.A$ contains $S.B$ with certainty, due to the referential integrity assumption. For $s(p) < 1$, Selinger et al. assume the chance that $R.A$ is still in $S.B$ reduces proportionally to the selectivity of p. For subqueries involving multiple predicates and relations, Selinger et al. express hope that the estimate remains accurate.

Thus, we conclude that in System R the selectivity estimate of a single predicate is accurate only when uniform distribution and uniform spread holds. On top of that, the selectivity of a combination of predicates is accurate when predicate independence holds. These assumptions are frequently violated. – Good for all researchers who contributed new cardinality estimation techniques since Selinger's 1979 paper [78].

Note that the approach is not complete. By complete we mean that, for an arbitrary relational algebra expression, a cardinality estimate can be derived. For completeness, we mention the following with respect to the system R estimator: Selection predicates of type A `like 'MA*'` and non-equi join predicates, e.g., $A < B$, are not considered. In addition, queries containing `GROUP BY` and `UNION` remain unmentioned. Also, projections and semijoins find no attention, which might be justified with respect to the cardinality of SQL queries. - Unless a query contains `DISTINCT`.

Chapter 4

State of the Art

In this section, existing approaches to cardinality estimation are discussed. Sections 4.1 to 4.4 cover the different classes of cardinality estimators. In Section 4.5, the shortcomings of existing approaches are discussed.

Note that some chapter-specific related work will also be discussed in the later sections of this thesis.

4.1 Sampling

In sampling, one observes a subset of a population to estimate something about the whole population. Sampling is versatile and used in many different sciences. An excellent overview of sampling techniques can be found in the book by Thompson [83].

The simplest sampling-based cardinality estimator is **Bernoulli sampling**, also known as **Simple Random Sampling**. To estimate the selectivity of a (complex) predicate p over a single relation R, we draw a sample $R' \subseteq R$. Each tuple in R' is drawn independently and with the same probability. Then, the selectivity of p is estimated as $\hat{s} = \frac{|\sigma_p(R')|}{|R'|}$. To obtain an estimate for the cardinality $|\sigma_p(R)|$, we simply compute $\hat{s} \cdot |R|$. For join size estimation using Bernoulli sampling, we first draw a sample R' from table R and a sample S' from table S. Then, the join selectivity is estimated as $\frac{|R' \bowtie_{p_J} S'|}{|R'| \cdot |S'|}$. In Bernoulli sampling, we essentially sample from the cross product, as illustrated in the following. Suppose R' is a 1% sample of R. Then we expect to have 1% of $R's$ join tuples. Further suppose S' is a 0.2% sample of S. Then we expect that each tuple in R' finds 0.2% of the matching tuples in S. Hence, in expectation, we only have a tiny $1\% \cdot 0.2\% = 0.002\%$ sample of the join. Thus, Bernoulli sampling is simply a $\frac{|R'|}{|R|} \frac{|S'|}{|S|}$ fraction of the cross product. A similar observation was made in the Aqua paper by Acharya et al. [10].

Correlated sampling by Vengerov et al. [86] is used for join size estimation and addresses the problem that Bernoulli sampling draws independent samples.

In correlated sampling, instead of sampling tuples independently, a hash function h that uniformly maps join attribute values to $[0, 1]$ is used to decide which tuples to sample. A sample R' with expected size $s_R \cdot |R|$ consists of all tuples whose join attribute value has a hash value less than some $s_R \in (0, 1]$. The sample of S' is obtained in the same fashion. An unbiased join size estimate is $\frac{1}{\min(s_R, s_S)}|R' \bowtie_{p_J} S'|$. For skewed data, the variance of this estimator is large, though. For instance, if one join attribute value occurs very frequently, then either all tuples with that value are in the sample or none. This effect is illustrated in the following example: Consider the case where 50 join attribute values in S are *heavy hitters* that occur very frequently and make a *large* contribution to the join size. Suppose we choose $s_S = 0.01$ to draw an expected 1% sample of S. This means that with probability roughly $(1 - 0.01)^{50} > 60\%$, none of those tuples are included in S', causing us to underestimate by a *large* factor. Chen and Yi made a similar observation [23].

In 2017, Chen and Yi presented **two-level sampling** for join size estimation [23], where a sample is drawn in two steps. The first step is similar to correlated sampling, the join attribute values of each tuple are hashed to $[0, 1]$. At least one tuple with hash value less than some $s_R \in (0, 1]$ is added to the sample R'. All other tuples with hash value less than s_R are added to the sample R' only with the *level two* probability q_R. Denote by $R'(v)$ the sample tuples with join attribute value v. Then the two-level join size estimate is

$$\frac{1}{\min(s_R, s_S)} \sum_{v \in R.K \cap S.B} \Big(\frac{|R'(v)| - 1}{q_R} + 1 \Big) \Big(\frac{|S'(v)| - 1}{q_S} + 1 \Big).$$

The $-1/+1$ compensates for the tuple that is sampled with probability s_R, whereas all other tuples are sampled with probability $s_R \cdot q_R$. The authors use a slightly different notation. Observe that for $q = 1$, two-level sampling and correlated sampling are the same. Chen and Yi point out that the *level one* probability s_R can differ per join attribute value. They argue that two-level sampling improves correlated sampling by reducing the variance. We agree. Then, Chen and Yi extend their argument for selection predicates: "*it is obvious that the variance can only become smaller under selection predicates. As pointed out in [86], this is a major advantage of sampling based approaches. [...] Essentially, we optimize for the worst case; when predicates are present, the variance can only be smaller*" [23]. While we agree with the first sentence, we consider it as misleading. The last sentence argues that the worst case occurs for queries without selection predicates, which is exactly the opposite of our opinion. To observe why the above statement by Chen and Yi is not valid, recall that the variance of an estimator $\widehat{J}$ for join size J is

$$Var(\widehat{J}) = E[(\widehat{J} - J)^2] \overset{discrete}{=} \sum_{i=1}^{n} p_i (\widehat{J}_i - J)^2$$

where the p_i's denote the probabilities of the n possible join size estimates $\widehat{J}_i$. The problem is that the statement by Chen and Yi ignores that the join size J differs depending on the selection predicates! As a result, one compares

(non-relative) differences from different entities of J (Comparing apples and oranges). This conflicts with the insights that justify the q-error (and equivalent error measures). To put it in other words: An estimate that is off by 100 from a join size of 2000 is acceptable, but not so for a join size of 20. The authors' first sentence is misleading since only the *relative variance* would have captured this. Huang et al. [39] published a technique that is similar to two-level sampling, but simpler. Wang and Chan propose to improve two-level sampling by learning a discrete distribution from a sample [88]. Note that correlated and two-level sampling draw samples for a specific join attribute. For joins on different attributes, new samples must be drawn.

4.2 Histograms

This section discusses histograms, in particular frequency histograms. A histogram is a data structure that approximates the frequency distribution of some domain D, e.g., the values of some attribute. To approximate the frequency distribution of D via histogram, D is split in n partitions. Each partition induces a *bucket*. A bucket contains the number of values in D that fall into that bucket's partition.

The number of tuples in a value range $Q := [q_1, q_2]$ (think of a range predicate) is at least the sum of the frequencies in the buckets that Q *contains* [80], and at most the sum of the frequencies in the buckets that Q *intersects* [80]. A bucket, with bucket range $P := [p_1, p_2]$, is contained by Q if P is a subinterval of Q. The bucket intersects Q if the intersection of P and Q is not empty.

Histograms differ in their partitioning scheme, and some also in their intra-bucket approximation scheme. Well-known partitioning schemes include Equi-width [48], Equi-depth [72], Max-diff and v-optimal [74], as well as q-optimal [63]. Different bucketing schemes induce different complexity for construction and when queried. For instance, for construction, the simple equi-width [48], requires only the minimum and maximum values in a column. Then, each of the n buckets has width $\frac{\max(T.A)-\min(T.A)+1}{n}$ and the histogram can be built in a single run over the data. The bucket corresponding to some value $v \in A$ can be accessed directly via its zero-based index $\left\lfloor \frac{val-\min(A)}{\max(A)-\min(A)+1} \cdot n_A \right\rfloor$.

The domain D on which a histogram is built can be multi-dimensional, i.e., D can consist of more than one attribute. Samet gives an overview over multi-dimensional histograms [76]. STHoles is a multi-dimensional histogram that is specifically designed for selectivity estimation [21]. Multi-column statistics for join size estimation must be built on the join result. IBM DB2 supports multi-column histograms with up to 100 buckets on *statistical views* [3].

4.3 Sketches

Sketches are small summaries of data. A sketch is constructed by a streaming algorithm, i.e., an algorithm that makes a fixed number of passes, usually one,

Sketch	Unary Op.	Binary Op.	Citation
HyperLogLog	Distinct count	Size of union/intersection	[31]
AKMV	Distinct count	Size of union/intersection/ set difference/join	[18]
AGMS	Self join size	Join size	[13]
Fast-AGMS	Self join size	Join size	[25]
KLL	Quantiles		[42]
Manku	Quantiles		[57]

Table 4.1: This table describes what value sketches approximate when either used in isolation or in combination with sketches for other attributes.

over the data. After construction, a sketch allows one to estimate some value, e.g., the distinct count of some attribute or some attribute's median value. In addition, some sketches can be combined with sketches for other attributes to obtain a different type of estimate. For instance, the combination of some sketches allows to obtain an estimate for the intersection size of two attributes or some join size. Table 4.1 lists, for a selection of sketches, the value the sketch estimates when used alone (*Unary Op.*) and when used in combination with a sketch for a different attribute (*Binary Op.*). In addition, the original paper of each sketch is cited.

In the context of this thesis, distinct count sketches and join size sketches matter. The distinct count sketches AKMV and HyperLogLog are described in the prerequisites, in Section 6. For distinct count sketches, we exploit a property that many sketches share: mergeability. Suppose we have n sketches for n partitions of some attribute A. Then mergeability states that one can merge the n sketches into one sketch that is the indistinguishable from a sketch that was built on all partitions of A directly.

To estimate join sizes, several sketches were invented. Rusu gives a good summary [75]. All join size sketches work in a similar fashion, as described in the following. To build a join size sketch, one iterates over an attribute and incrementally updates a matrix of counters. Each row in the matrix approximates the frequency distribution of that attribute. To estimate the size of a join, one multiplies two sketches. Sketch multiplication is defined as the row-wise inner product of counter matrices. The result is a row of numbers, of which one serves as the join size estimate. The choice depends on the sketch and how its counters are filled. The AGMS and Fast-AGMS sketch take the median [13, 25], whereas the Count-Min sketch takes the minimum [27]. Unlike samples or histograms, sketches cannot incorporate predicates. Since each sketch can be large (many kB or MB), storing many join size sketches per relation, to merge them as needed, may not be admissible. Recently, Izenov et al. published a notable approach [41] to estimate the result cardinalities of queries with multiple joins over attributes of different domains. Their approach relies on fast-AGMS sketches and a technique called *sketch merging*. Cai et al. [22] presented an interesting approach to estimate upper bounds for join sizes but, as they note in Appendix C, query

optimization time exceeds query execution time and takes more than an hour for their benchmark queries.

4.4 Machine Learning

Since recent years, it became popular to apply machine learning techniques to cardinality estimation. Simply speaking, these techniques learn a mapping from query to its result cardinality. The learning step, also referred to as *training*, usually requires a training set of queries and their corresponding result cardinalities. All machine leaning estimators have in common that they do not consume a query, e.g., a SQL string, directly. Instead, a numerical representation of a query, called *feature vector*, is consumed. A function that maps a query to its feature vector is called *query featurization technique* (QFT).

Kipf et al. featurize queries into different sets and learn their cardinalities with a specific Multi Set Convolutional Network (MSCN) architecture [47]. This approach supports both base table and join size estimates with multiple predicates. However, its QFT is lacking domain knowledge and explainability, since it learns an implicit black box featurization through its structure during training.

Woltmann et al. [93] extend the earlier approach [47] to handle arbitrary subsets of a database schema by training several local models. While this approach reduces the disadvantages of Kipf et al., this work only examines one type of QFT for one predicate per attribute.

Yang et al. present Naru [94]. They use autoregressive models for learning the conditional joint probability for point queries. This introduces computational overhead to estimate range queries, since their estimate is the sum over multiple point queries. Additionally, the order of attributes needs to be fixed, which makes generalization difficult. The authors illustrate the time and memory costs very well. One epoch of training, using their data, takes 50-75 seconds. After training, around 100 estimates can be produced per second, each estimate uses the full power of an NVIDIA V100 GPU. Naru uses a very simple QFT that also only allows one predicate per attribute. The authors do not present details about its impact on the models' quality. A similar approach by Hasan et al. [35], who acknowledge the impact of query featurization on the models' quality but do not further research in this direction.

Dutt et al. [28] present similar approaches to [47, 93], but focus on the estimator models' complexity, like memory footprint and training time, being as small as possible. This work supports gradient boosting as a lightweight model architecture but does not detail the impact of QFTs.

Hilprecht et al. present an approach [38] to learn a model directly from the data, without the need for a training set of queries. It uses Sum-Product-Networks (SPN) to model the attribute value distributions of each base table. For multi-predicate queries over joins, the outputs of the base table models are combined. The approach relies to some extent on sampling for finding matching join attributes for the construction of SPN. Similar to Kipf et al., DeepDB implicitly learns a black box featurization for queries.

4.5 Shortcomings

In the previous sections, we have discussed existing approaches to cardinality estimation. Some of the shortcomings of existing cardinality estimation techniques were already outlined in the previous sections. This section concisely summarizes the shortcomings of current techniques. Thus, it demonstrates where there is potential to improve the current state.

1. **Sampling**

   - For selective selection predicates, no tuple in a sample may qualify. This results in bad estimates for the number of qualifying tuples in a table. This problem was recognized, and attempted to solve, before [37].
   - For joins, joining two Bernoulli samples usually results in an empty join result, even without selection predicates.
   - For correlated sampling and two-level sampling for join size estimation, selective predicates may still cause bad estimates. In addition, the samples are drawn for specific join attributes. This might result in many samples per table. Furthermore, as Chen and Yi point out [23], for multi-table joins, the sample may be specific for a given query graph shape.

2. **Histograms**

   - Consider query 6 of the TPC-H benchmark:

     ```
     SELECT sum(l_extendedprice * l_discount) as revenue
     FROM lineitem
     WHERE
       l_shipdate >= date '1994-01-01'
       AND l_shipdate < date '1995-01-01'
       AND l_discount between 0.06 - 0.01 AND 0.06 + 0.01
       AND l_quantity < 24
     ```

     While a database system may maintain histograms for each of the attributes `l_shipdate`, `l_discount`, and `l_quantity`, it is unlikely that a 3-dimensional histogram for these three attributes is maintained. It is not trivial how to combine the knowledge from these three estimates to obtain an estimate for the result cardinality of the above query. In particular, all existing approaches fall short when there additionally exists a sample, and we want to use both the histograms and the sample to obtain a consistent estimate.

3. **Join Size Sketches**

   - While join size sketches are artful data structures with some provable statistical properties, their application in query optimization is limited. As we have noted in the introduction, in the context of this

thesis, only queries with selection predicates matter. However, current join size sketches cannot incorporate predicates after they are constructed.

4. **Machine Learning**

   - Many machine learning techniques are not as versatile as commonly believed. In particular, no machine learning model for cardinality estimation operates on a SQL string. Instead, a numerical representation of the query serves as input to the model. The class of queries that can be transformed to a numerical representation is often very limited, e.g., only one selection predicate per attribute [35]. In addition, the type of selection predicates are limited. For instance, predicates on strings as `like %foo` are usually not supported.
   - Considering that multiple machine learning models are needed for the data stored in a database system, their time and space consumption is very large for a typical statistics catalog.
   - The number of training queries needed to obtain good estimates on unseen queries is very high. Obtaining the training queries usually comes at a significant cost, as was noted in [38]. Either, training queries are generated, which has high computational cost. Or, they are obtained from query logs, but then we must wait for the query log to contain a sufficient number of representative queries.
   - After all, the accuracy of (current) machine learning models for cardinality estimation is not very convincing [47].

Our new cardinality estimation techniques, presented in the following sections, are designed to address some of the above shortcomings and improve the state of the art in cardinality estimation.

Chapter 5

Selection Predicates over Single Relations

5.1 Introduction

This chapter presents an approach to estimate the selectivity of multiple predicates over a single relation. Selectivities are estimated by available information. Many database systems maintain synopses, e.g., histograms, and, in addition, provide sampling facilities. The crux of the matter is how to utilize this information: Consider a query with predicates $p_1 \equiv$ `A > 5`, $p_2 \equiv$ `B between 2.7 and 3.5`, and $p_3 \equiv$ `C = 'green'` over a relation R with attributes A, B and C. Suppose the system provides histograms over the single attributes that give approximate selectivities for p_1, p_2, and p_3 as well as a multi-dimensional histogram that approximates the joint distribution of B and C and, thus, gives an approximate selectivity for $p_2 \wedge p_3$. Furthermore, assume that the system maintains a sample of R. One possibility of finding estimates for the unknown selectivities of $p_1 \wedge p_2$ and $p_1 \wedge p_2 \wedge p_3$ is to simply compute the ratio of qualifying entries in the sample. However, if R is large and the selectivities of the predicates are low, the quality of this estimate is often insufficient. Another approach is to derive estimates solely based on the known synopses. Elaborate methods based on the principle of maximum entropy have been developed by Markl et al. to consistently process multi-dimensional synopses [58]. The question remains how to utilize both synopses and sampling to estimate selectivities. To the best of our knowledge, Yu, Koudas and Zuzarte are the only ones who have attempted to answer this question [95]. As the main problem, they consider selectivities obtained from sampling that are inconsistent with selectivities obtained from synopses, e.g., the selectivity of $p_1 \wedge p_2$ derived from a sample may be larger than the selectivity of p_1 as provided by a histogram, regardless of the fact that both cannot hold true simultaneously. Hence, their idea is to refine estimates derived from sampling until they comply with the selectivities

known from synopses. However, as we will see later, their approach has limited capabilities.

In this chapter, we introduce CSE, a novel approach to selectivity estimation for conjunctive queries for single relations, that consistently combines sampling and synopses. To this end, for each selectivity derived from a sample and each selectivity obtained from a synopses structure, we construct intervals that contain the true selectivity either guaranteed or with high probability. We then go on to produce selectivity estimates by solving an optimization problem that is constrained to these intervals. A key property of our approach is that it can incorporate multi-dimensional synopses. Moreover, our approach proves robust in our evaluation in all scenarios. In particular, our approach not only outperforms existing state-of-the-art methods in a scenario where the selectivities of some of the predicates found in a query are known precisely, but also performs best in real-world scenarios where synopses provide selectivities with approximation errors. This is due to our method that extracts intervals, instead of point-estimates, from sampling and synopses to overcome the issue of estimating selectivities based on inaccurate approximations.

The remainder of the chapter is structured as follows: In the next section, we discuss some preliminaries and introduce our notation. We then present related work in Section 5.3. In Section 5.4, we introduce our CSE approach. Section 5.5 contains an extensive evaluation of our approach in terms of prediction accuracy and run time in comparison to other state-of-the-art approaches under scenarios with varying parameters. To the best of our knowledge, we are the first who consider synopses with approximation errors in our evaluation, as it is the case in real-world scenarios. Finally, we summarize the contents of this chapter.

5.2 Preliminaries

In this section, we introduce the notation used in this chapter and discuss preliminaries. A conjunctive query P, defined as a conjunction of n *simple predicates* or *boolean factors*, over a relation R represents the starting point of our discussion.

$$P := p_1 \wedge p_2 \wedge \cdots \wedge p_n$$

A predicate is simple if it compares an attribute value to a literal. We denote by $N = \{1, \ldots, n\}$ the index set of P.

5.2.1 Predicates and Selectivities

A selectivity is a value in the interval $[0, 1]$ and is defined as the fraction of entries in a data set or relation that satisfies some specified predicate. We distinguish selectivities induced by predicates that are defined by two formulae. For both, the argument X is a subset of the index set N of a given conjunctive query P, i.e., $X \subseteq N$. The first formula is defined as

$$F_\beta(X) := \bigwedge_{i \in X} p_i,$$

i.e., $F_\beta(X)$ is a conjunction of those predicates whose index is contained in X. In case $X = \emptyset$, we define $F_\beta(X) \equiv true$.

The second formula is defined as

$$F_\gamma(X) := \bigwedge_{i \in X} p_i \wedge \bigwedge_{i \in N \setminus X} \neg p_i,$$

i.e., $F_\gamma(X)$ defines the *minterms* of the conjunctive query P. For a boolean function of n variables, a minterm is defined as a conjunction in which each of the n variables appears exactly once, possibly in its complement form.

To illustrate the difference in the formulae and their consequence on the selectivity, consider the conjunctive query $p_1 \wedge p_2 \wedge p_3$ with index set $N = \{1, 2, 3\}$. Let $X = \{1, 3\} \subseteq N$. Then $\beta(X)$ is the selectivity of $F_\beta(X) = p_1 \wedge p_3$, which we call the β-selectivity of X. Similarly, $\gamma(X)$ is the selectivity of $F_\gamma(X) = p_1 \wedge p_3 \wedge \neg p_2$, which we call the γ-selectivity of X.

Note that for all X, the β-selectivity $\beta(X)$ is greater than or equal to the γ-selectivity $\gamma(X)$. This is because $F_\gamma(X)$ contains at least the predicates in $F_\beta(X)$, and additional predicates imply a lower or at most unchanged selectivity. For the same reason, we have that $\beta(X') \geq \beta(X)$ for all $X, X' \subseteq N$ with $X \supset X'$. Furthermore, note that from our definition above $F_\beta(\emptyset) \equiv true$, it follows that $\beta(\emptyset) = 1$ since every entry in a data set or relation satisfies this condition.

Observe that both F_β and F_γ depend only on $X \subseteq N$. Since all $X \subseteq N$ form the power set of N, which is known to contain 2^n elements, the number of β- and γ-selectivities is 2^n each.

Finally, observe that all $X \subseteq N$ can be numbered by bitvectors $bv(X) := (d_n, \ldots, d_1)$, where $d_i = 1$ if $i \in X$, and $d_i = 0$ otherwise, for $1 \leq i \leq n$. Therefore, without introducing ambiguity, we refer to a formula or selectivity likewise by its characteristic bitvector $bv(X)$ for some set of indices X.

5.2.2 Relation between β- and γ-selectivities

Every conjunctive query $F_\beta(X)$ can be expressed as the disjunction of those minterms $F_\gamma(Y)$ that positively contain at least the literals in $F_\beta(X)$ [30, Chapt. 3.9]. For instance, for conjunctive query $p_1 \wedge p_2$ and $F_\beta(\{1\}) = p_1$, the minterms that positively contain at least the literals in $F_\beta(\{1\})$ are $F_\gamma(\{1\}) = p_1 \wedge \neg p_2$ and $F_\gamma(\{1, 2\}) = p_1 \wedge p_2$ and, thus, we have that $p_1 \equiv (p_1 \wedge \neg p_2) \vee (p_1 \wedge p_2)$.

It follows that every β-selectivity $\beta(X), X \subseteq N$ can be computed from γ-selectivities as

$$\beta(X) = \sum_{X \subseteq Y \subseteq N} \gamma(Y), \tag{5.1}$$

or in words: $\beta(X)$ is composed of those $\gamma(Y)$ where at least the predicates contained in $F_\beta(X)$ occur positively in $F_\gamma(Y)$. Figure 5.1 illustrates the relationship between β-selectivities and γ-selectivities according to Equation 5.1 for the previous example $p_1 \wedge p_2$.

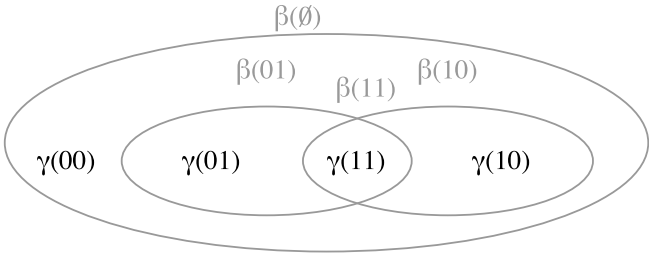


Figure 5.1: Grey ellipses mark the γ-selectivities that contribute to each β-selectivity for the conjunctive query $p_1 \wedge p_2$.

Note that we can compute all $X \subseteq Y \subseteq N$ efficiently by considering bitvector $bv(X)$ and enumerating all bitvectors $bv(Y)$ that contain a 1 at least at those positions where $bv(X)$ contains a 1 since

$$\{Y|N \supseteq Y \supseteq X\} \iff \{Y|bv(X) = bv(X)\&bv(Y)\},$$

where & denotes *bitwise AND*. Observe that $\beta(\emptyset)$ is the sum of all complete conjuncts, since all $Y \supseteq \emptyset$.

As an example, consider the conjunctive query $p_1 \wedge p_2$ and let $X = \{1\} \widehat{=} 01$. Then the set of all $Y \supseteq X$ is $\{\{1\}, \{1,2\}\} \widehat{=} \{01, 11\}$ and $\beta(\{1\}) = \gamma(01) + \gamma(11)$.

5.2.3 Matrix Representation

Since for all $X \subseteq N$ Equation 5.1 gives one linear equation, together these equations form a system of linear equations $b = Cx$ with vectors $b = (\beta(\emptyset), \ldots, \beta(N))^T$ and $x = (\gamma(\emptyset), \ldots, \gamma(N))^T$, where T as superscript denotes *transposed*, and design matrix C, to which we refer as the *complete design matrix*. For zero-based indexing, the definition of the $2^n \times 2^n$ matrix C follows directly from the enumeration of summands in Equation 5.1:

$$C_{i,j} = \begin{cases} 1 & \text{if } bv(i) \subseteq bv(j) \\ 0 & \text{else} \end{cases} \tag{5.2}$$

Note that C is Boolean and each row indicates which γ-selectivities contribute to a β-selectivity. Furthermore, note that we assume the $\beta(X)$ in b and the $\gamma(X)$ in x to be sorted in ascending order of their bitvector-value $bv(X)$.

Consider the example system $Cx = b$ for $p_1 \wedge p_2$:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \gamma(00) \\ \gamma(01) \\ \gamma(10) \\ \gamma(11) \end{pmatrix} = \begin{pmatrix} \beta(00) \\ \beta(01) \\ \beta(10) \\ \beta(11) \end{pmatrix}.$$

Given all γ-selectivities, compute all β-selectivities as Cx. In case that we are given all β-selectivities, we (conceptually) compute all γ-selectivities by inverting C and computing $C^{-1}b$.

Note that it is easy to see that the matrix C is indeed invertible by observing that it is upper-triangular with no zeros on the main diagonal.

5.2.4 A Linear System Induced by Synopses

Synopses structures, like histograms, sketches or wavelets, approximate the distribution of single attributes or attribute groups. In terms of our notation, synopses structures provide β-selectivities. Ideally, a database system would maintain synopses over all possible combinations of attributes in the database. Selectivity estimation would be easy then, since for every conjunctive query P with index set N, we could simply obtain the selectivity of P, i.e., $\beta(N)$, from the synopses structures. Additionally, we could obtain the selectivity $\beta(X)$ for all $X \subseteq N$ and formulate the linear system $b = Cx$ that we saw in the previous section. Unfortunately though, since the number of attribute combinations grows exponentially in the number of attributes, it is infeasible to maintain synopses for all attribute groups. In reality, synopses are only available for low-dimensional attribute groups. For instance, a database system may maintain single attribute statistics, referred to as 1D synopses, and statistics for attribute combinations of two attributes, referred to as 2D synopses. In this section, we derive a system of equations similar to that in the previous section, but this time induced by the synopses maintained in a database system.

As before, let N be the index set of a given conjunctive query P. Suppose we know, due to synopses, the β-selectivities $\beta(X)$ for some but not all $X \subseteq N$. We collect these X in a set G, to which we refer as the *knowledge set*, since it specifies the β-selectivities that are known from synopses. Denote by b the corresponding $|G|$-dimensional vector of β-selectivities $\beta(X), X \in G$ to which we refer as the known selectivities. Each β-selectivity in b induces a linear equation defined by Equation 5.1. Together these equations form the linear system $b = Ax$, where $x = (\gamma(\emptyset), \ldots, \gamma(N))^T$ holds all γ-selectivities and A is a $|G| \times 2^n$ design matrix defined as follows

$$A_{i,j} = \begin{cases} 1 & \text{if } bv(G_i) \subseteq bv(j), \\ 0 & \text{else.} \end{cases}$$

We refer to A as the *partial design matrix*. Note that every partial design matrix A is simply the selection of those rows in the complete design matrix C that correspond to equations for which the $\beta(X)$ is known. Furthermore, note that for the linear system $b = Ax$ we assume the $\beta(X)$ in b, the $\gamma(X)$ in x, and the sets of indices X in the knowledge set G to be sorted in ascending order by their bitvector-value $bv(X)$. Finally note that, unless all $X \subseteq N$ are part of the knowledge set G, the linear system $Ax = b$ is underdetermined. Thus, assuming $Ax = b$ is solvable, there exist infinitely many solutions for x.

As an example, consider the conjunctive query $p_1 \wedge p_2 \wedge p_3$ with index set $N = \{1, 2, 3\}$ over the attributes A, B, C of some relation R. Let $p_1 \equiv$ `A between 1 and 10` with a selectivity of 0.1, $p_2 \equiv$ `B` $\leq$ `100` with a selectivity of 0.2 and $p_3 \equiv$ `C = 5` with a selectivity of 0.01. Assume the database

system maintains statistics for each individual attribute and multivariate statistics for the attribute group A, B. Then, the knowledge set is $G = \{\{\emptyset\}, \{1\}, \{2\}, \{1,2\}, \{3\}\} \widehat{=} \{000, 001, 010, 011, 100\}$. The β-selectivity of each simple predicate is already given above. For $p_1 \wedge p_2$ suppose a selectivity of 0.05; note that it would be 0.02 if p_1 and p_2 were independent. Then, the system of linear equations $Ax = b$ is

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} \gamma(000) \\ \gamma(001) \\ \gamma(010) \\ \gamma(011) \\ \gamma(100) \\ \gamma(101) \\ \gamma(110) \\ \gamma(111) \end{pmatrix} = \begin{pmatrix} 1 \\ 0.1 \\ 0.2 \\ 0.05 \\ 0.01 \end{pmatrix},$$

where solutions for vector x, i.e., $(\gamma(000), \gamma(001), \ldots, \gamma(111))^T$ are admissible if all γ-selectivities lay in $[0, 1]$.

In general, the linear system $b = Ax$ captures all knowledge that is available due to synopses in a database system. In [58] the linear system $b = Ax$ is given in implicit form, where Markl et al. substitute the design matrix A by so-called *components* induced by the elements of the knowledge set.

Note that by now we have assumed that synopses structures allow us to *know* some β-selectivity. This is not correct. Synopses structures *approximate* β-selectivities, and we will see the impact of this distinction in later sections.

5.3 Existing Approaches

In this section, we discuss existing approaches that relate closely to this chapter.

Recall that System R [78] assumes predicate independence. A novel heuristic by Microsoft [9] takes statistical relationships into account by assuming that there naturally exist similar statistical relationships among attributes. Since database systems usually store statistics for at least all single columns, the β-selectivity for each of the n simple predicates in a query can be extracted. In a next step, these selectivities are sorted in ascending order, i.e., $\beta(1) \leq \beta(2) \leq \cdots \leq \beta(n)$. The core idea is then to take the first k selectivities and compute an estimate as $\prod_{i=1}^{k} \beta(i)^{0.5^{(i-1)}}$. Note that this implies that $n - k$ selectivities do not contribute to the estimate. However, since $\beta(i+1) \geq \beta(i)$ and due to the *exponential back-off*, the factor $\beta(i)^{0.5^{(i-1)}}$ converges to the limit 1, where it does not change to the product and, therefore, to the estimate anyway. The Microsoft SQL Server Team choses $k = 4$ and justifies their choice with the rapid speed of convergence. To give some intuition for this choice of k, note that for an estimate computed with $k = 4$ to differ from an estimate computed with $k = 5$ by 50%, i.e., $s_5^{1/16} = 0.5$, the fifth most selective selectivity s_5 must have a value of around 0.00002. In their work on synopses-based

HASE(b, R')

Input: known selectivities b,
sample R'

Output: estimated selectivity for conjunctive query P

1 let $\{w_j, j \in R'\}$ be a set of variable weights
2 let $\{I_j, j \in R'\}$ be a set of indicator vectors, cf. text.
3 let c be a constant dampening vector
4 $\underset{w}{\text{minimize}} \quad cD(w)$
subject to $\quad \frac{1}{m} \sum_{j \in R'} w_j \cdot I_j = b$
5 **return** $\frac{1}{m} \sum_{j \in \{I_j = \mathbf{1} | j \in R'\}} w_j$

consistent selectivity estimation via maximum entropy [58], Markl et al. show how to exploit all available synopses to obtain more accurate estimates. Their method is based on the principle of maximum entropy. Thus, in absence of knowledge, their estimator assumes independence. One benefit of maximum entropy is its interpretability: When estimates are bad, more information on attribute combinations not satisfying the independence assumption is needed. To find the desired selectivities, Markl et al. formulate an optimization problem. The objective is to find the γ-selectivities that maximize the entropy subject to the constraints given by the system of equations $Ax = b$ that we have introduced in Section 5.2.4. However, in case that $Ax = b$ is unsolvable, there exists no solution to the optimization problem. $Ax = b$ is unsolvable if the β-selectivities in b are mutually inconsistent, which means that not all selectivities in b can hold true at the same time. Such inconsistencies occur because synopses structures usually only yield approximations for selectivities. To ensure that $Ax = b$ is solvable, Markl et al. adjust the β-selectivities in b when necessary. The adjustments are computed in a minimal way with respect to a metric. Different approaches exist in the literature to compute minimal adjustments of b, based on l_1-norm [58] and based on l_q-norm [65]. Once a solution for the vector of γ-selectivities x is found, Equation 5.1 is applied to obtain estimates for the unknown β-selectivities.

To the best of our knowledge, Yu et al., in their *HASE* paper [95], were the first and only ones who combined selectivities from sampling and synopses. The core idea is to introduce variables that compensate for differences between sampling and synopses. Then, find the smallest compensation factors possible. In the following, we describe in a nutshell how HASE works. Note that we have reformulated and simplified their problem specification to make it consistent with our notation: As before, let N denote the index set of a given conjunctive query P. Assume, due to some synopses, we are given a $1+|N|$-dimensional vector b of β-selectivities containing the β-selectivity when no predicate is applied, i.e., 1, and the β-selectivity of each simple predicate in the given conjunctive query. b is regarded to be a vector of true selectivities. In addition, suppose we have a

random sample $R', |R'|=m$ from the table under consideration $R, |R|=M$. Associated with each sample tuple $j \in R'$ is a $1+|N|$-dimensional *indicator vector* I_j that indicates which simple predicates a sample tuple qualifies. In particular, we set $I_{j0}=1$ for all sample tuples, and $I_{ji}=1, 1 \leq i \leq |N|$ if sample tuple j satisfies p_i and $I_{ji}=0$ otherwise. Note that those sample tuples that satisfy the conjunctive query P have $I_j = \mathbf{1}$, i.e., all entries are set to one.

Further note that, based on the sample, the estimate for the b is $b_{\text{smpl}} = \frac{1}{m}\sum_{j \in R'} I_j$. However, due to the imprecise nature of sampling, we expect the sampling-selectivities to be inconsistent with the synopses-selectivities, i.e., $b_{\text{smpl}} \neq b$ Hence, Yu et al. associate a weight w_j with each sample tuple $j \in R'$. Then, an admissible solution satisfies $\frac{1}{m}\sum_{j \in R'} w_j \cdot I_j = b$. In general, there exist infinitely many of such assignments for the weights. The objective is to find the one with a minimal sum of (mapped) weights. The weight vector w is mapped using a distance function D and a dampening vector c, that associates a user-defined dampening factor with each component in w. Then, the final selectivity estimate for the conjunctive query P is $\frac{1}{m}\sum_{j \in \{I_j=\mathbf{1} | j \in R'\}} w_j$, i.e., the the sum over the weights of those tuples that qualify the conjunctive query P, normalized by the sample size m. A codification of this process is given in Algorithm HASE. Note that we have simplified the problem statement of Yu et al. by assuming each tuple from R is included in the sample R' with equal probability.

The limitation of HASE is that they can handle 1-dimensional synopses only. A generalization to multi-dimensional synopses introduces potential mutual inconsistencies, however they do not consider methods to overcome inconsistencies in the known selectivities. In addition, as we will see in our evaluation, HASE fails at exploiting the potential of combining sampling and synopses in terms of accuracy.

5.4 Combined selectivity estimation

In this section, we present CSE, a novel technique to estimate the selectivities for some conjunctive query P. Section 5.4.1 demonstrates how to construct sampling bounds by deriving confidence intervals for all γ-selectivities associated with P. Section 5.4.2 shows how to derive bounds on β-selectivities from synopses. In Section 5.4.3 we show how to formulate a constrained optimization problem where the constraints are given by the bounds obtained in Sections 5.4.1 and 5.4.2. The optimal solution to this optimization problem serves as the selectivity estimate. One approach to compute the optimal solution is presented in Section 5.4.4.

5.4.1 Sampling Bounds

Sampling allows to estimate the selectivity of any type of predicate. It is well-known that an unbiased estimate of the selectivity of some conjunctive query can be obtained by counting the number of qualifying samples and dividing it by

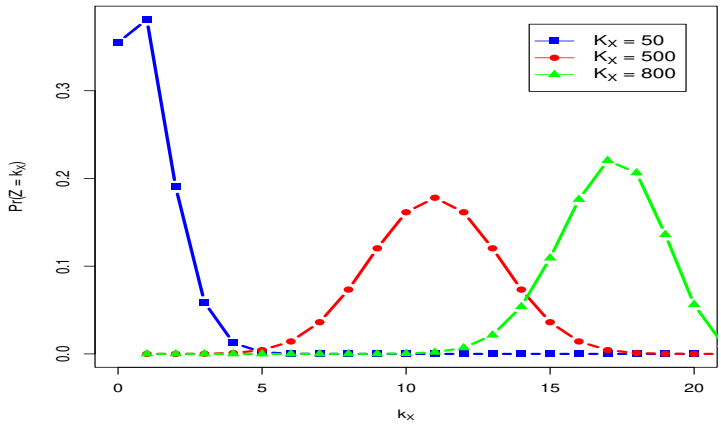


Figure 5.2: Hypergeometric distributions with parameters M=1000, m=20, and $K_X \in \{50, 500, 800\}$.

the sample size. However, if the number of qualifying samples is low, the quality of the estimate is often insufficient. Here we want to investigate a method that uses a sample to construct confidence intervals for γ-selectivities such that the true γ-selectivity is contained in the interval with high probability.

Assume we draw a random sample of size m from a population of size M. Say we observe that k_X items in the sample satisfy the predicate $F_\gamma(X)$ corresponding to some $X \subseteq N$, where N refers to the index set of some conjunctive query. The goal is to determine K_X, the number of items in the population that qualify, since $K_X/M = \gamma(X)$. Clearly, using M, m, k_X, we cannot find an approximation for K_X that is guaranteed to be correct.

However, we can bring certainty, to an arbitrary degree, to sampling by constructing confidence intervals $[\gamma^l(X), \gamma^u(X)]$ with high confidence levels. Later, we use these confidence intervals to estimate unknown β-selectivities.

We model sampling as an *urn problem* with the following characteristics: (1) each item either qualifies or does not qualify, (2) we draw without replacement. This urn problem induces the *hypergeometric distribution* [51, Chapt. 3.2]

$$\Pr(Z = k_X) = \frac{\binom{K_X}{k_X}\binom{M-K_X}{m-k_X}}{\binom{M}{m}},$$

where $\Pr(Z = k_X)$ denotes the probability of drawing exactly k_X items that qualify in m draws. Figure 5.2 illustrates the hypergeometric distribution function for M=1000, m=20, and $K_X \in \{50, 500, 800\}$. Note that the hypergeometric distribution is a discrete distribution; the lines between the points are only drawn to better illustrate its shape. Observe that, unlike the normal distribution, the hypergeometric distribution is unsymmetrical and bounded. In addition, its shape heavily depends on K_X.

# qual. items	lower bound	upper bound
0	0	11681.2
1	21.1726	13516.3
2	164.683	15254.2
3	405.334	16921.9
4	716.157	18535.9
5	1080.77	20106.9
6	1488.34	21642.6
7	1931.26	23148.5
8	2403.93	24628.8
9	2902.12	26087
10	3422.51	27525.7
20	9438.11	41177.3
30	16316.4	54025.3
40	23677.9	66411.7
50	31362.3	78486.2
100	72512.8	136193
200	161231	245273
500	447753	552247
1000	988319	1000000

Table 5.1: Wilson Score interval bounds for a population of size 1M and a sample of size 1k.

Recall that our goal is to construct a confidence interval for K_X. Hence, we must determine the pair of random variables lower-K_X, denoted by K_X^l, and upper-K_X, denoted by K_X^u, such that

$$\Pr(K_X^l \leq K_X \leq K_X^u) = 1 - \alpha,$$

where $\alpha \in (0, 1)$. Then, $[K_X^l, K_X^u]$ is a confidence interval for K_X with confidence level $1 - \alpha$. We experimentally found $\alpha = 10^{-3}$ to be a good value. Therefore, we are quite certain the true cardinality K_X lays in the computed bounds.

Unfortunately, exact methods are computationally expensive [82]. However, assuming $m \ll M$, the hypergeometric distribution coincides with the binomial distribution[1], where the parameters are the sample size m and the success probability $\frac{K_X}{M}$. Indeed, a plot of the binomial distribution with parameters m=20 and $K_X \in \{\frac{50}{M}, \frac{500}{M}, \frac{800}{M}\}$ is practically indistinguishable from Figure 5.2, which is why we do not show the plot.

Due to the shape of the binomial distribution, it is not univocal how to derive a confidence interval for

$$\left[\frac{K_X^l}{M}, \frac{K_X^u}{M}\right] =: [\gamma^l(X), \gamma^u(X)].$$

Hence, several methods have been invented, e.g. Wilson Score interval [90], Jeffreys interval [20], and Clopper–Pearson interval [24]. We found the Wilson

[1] which refers to sampling with replacement

GetSamplingBounds(P, n, N, S, m, α)

Input: conjunctive query P with n predicates and index set N,
sample S of size m,
significance level α.
Output: lower bounds for all γ-selectivities in vector x^l.
upper bounds for all γ-selectivities in vector x^u.

```
1  let counts be an associative array
2  for each t ∈ S
3      let X be a bitvector of size n
4      for i = 0 to n
5          if p_i(t)
6              X[i] = 1
7          else
8              X[i] = 0
9      counts[X] = counts[X] + 1
10 z = QuantileStandardNormalDist(1 − α/2)
11 let x^l and x^u be associative arrays of size 2^n
12 for each X ⊆ N
13     k_X = counts[bv(X)]
14     T = z·sqrt(z^2 − 1/m + 4k_X(1 − k_X/m) + (4k_X/m − 2)) + 1
15     x^l[X] = (2k_X + z^2 − T)/(2(m + z^2))
16     x^u[X] = (2k_X + z^2 + T)/(2(m + z^2))
17 return (x^l, x^u)
```

Score interval method with continuity correction [69] to be a good method. The method is derived by the Yate's chi-squared test, that is used to test how likely it is that differences in observations occur by chance. The interval boundaries are efficiently computed as

$$\left[\frac{2k_X + z^2 - T}{2(m + z^2)}, \frac{2k_X + z^2 + T}{2(m + z^2)}\right], \tag{5.3}$$

where $T = z\sqrt{z^2 - \frac{1}{m} + 4k_X(1 - \frac{k_X}{m}) + (4\frac{k_X}{m} - 2)} + 1$, and z denotes the $1 - \frac{\alpha}{2}$ quantile of a standard normal distribution. However, if no sample items qualify, the lower bound must be taken as 0. Similarly, if all items qualify, the upper bound must be taken as 1.

Table 5.1 illustrates how the interval bounds close in in the number of qualifying items for a population of size 10^6, a sample of size 10^3 and a confidence level $1 - \alpha$ of 0.999.

Algorithm GetSamplingBounds describes how to compute $[\gamma^l(X), \gamma^u(X)]$ for all $X \subseteq N$. The result is stored in two vectors x^l and x^u for which it holds that

$$x^l \leq x \leq x^u, \tag{5.4}$$

with high probability. The algorithm first computes k_X for all $X \subseteq N$ in a single pass over the sample. Then, Equation 5.3 is applied to compute lower and upper bounds given a significance level α. The input is the conjunctive query P with n simple predicates and index set N, a sample S of size m and significance level α.

A notable property is that if the confidence interval is wide for some X it must be tighter for others, since each item in the sample must qualify the γ-predicate corresponding to some $X \subseteq N$. To see that, recall the discussion of minterms in Section 5.2.

5.4.2 Synopses Bounds

We have seen in Section 5.2.4 that synopses structures provide β-selectivities for certain single attributes or attribute groups. As mentioned in the related work section, provided β-selectivities are usually approximations of the true β-selectivity of some predicate. As such, they are subject to approximation errors. Approximation errors occur since synopses like histograms approximate selectivities based on frequencies and boundaries of buckets as well as assumptions regarding the distribution of values in histogram buckets. This can cause a system of equations induced by synopses $Ax = b$ to be unsolvable. As mentioned in the related work section, in such cases Markl et al. propose adjustments of the β-selectivities in the vector b to make $Ax = b$ solvable. However, adjusting selectivities adds a time-consuming step to the selectivity estimation process, cf. [58], where adjusting took longer than estimation.

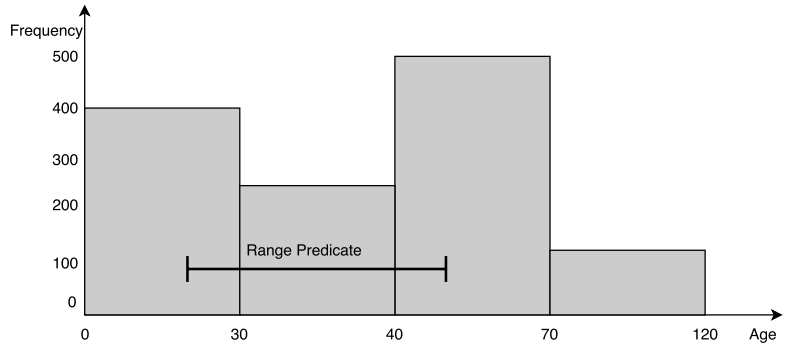


Figure 5.3: Example histogram with an example range predicate.

Here, we investigate an approach that relies on boundaries of histogram buckets. Figure 5.3 shows an example histogram that approximates the distribution of an attribute `Age`. The bar ranging from the first to the third bucket represents a range predicate. Clearly, since we have a histogram for `Age`, the index of this predicate would be part of the knowledge set G. However, the only thing we actually know is that the frequency of the second bucket is part of the result frequency, since the range of the predicate spans further than the bucket.

The frequencies of the first and third bucket can only be approximated via an intra bucket approximation scheme, that is, e.g., simply assuming an uniform distribution. Such assumptions can translate to wrong approximations, which ultimately lead to unsolvable systems of equations induced by synopses $Ax = b$.

Hence, we use bucket boundaries to derive an interval that contains the true frequency. In our example, this interval ranges from the frequency of the second bucket to the cumulated frequencies of the first three buckets. In general, the lower bound of the interval is the sum of the frequencies of all buckets contained by some predicate p. The upper bound of the interval is the sum of the frequencies of all buckets intersected by p. Note that a selectivity is simply the relative counterpart of a frequency. Therefore, for a given conjunctive query P with index set N, we can derive a lower bound $\beta^l(X)$ and an upper bound $\beta^u(X)$ for each X in the knowledge set G.

The bounds depend on the bucketing scheme of a histogram. In commercial database systems, histograms with different bucketing schemes can be found. Oracle uses top frequency histograms and equi-depth histograms. By default, the maximum number of buckets is 254 [2]. In IBM DB2, they employ equi-depth histograms with a maximum of 100 buckets [4]. In Microsoft SQL Server or Azure SQL Database, respectively, maxdiff histograms with a maximum of 200 buckets are used [7]. For multi-dimensional synopses, we recommend histograms with tight bounds, e.g., [80].

SAP HANA uses maxdiff histograms and q-optimal histograms, i.e., histograms with a maximum multiplicative error for estimates [5]. Q-optimal histograms give intra bucket guarantees that allow one to specify the width of the bounds upon histogram construction. Hence, given an obtained frequency, the bounds are already known.

The obtained lower bound $\beta^l(X)$ and upper bound $\beta^u(X)$ for each X in G then allow us to formulate the system of inequalities

$$b^l \leq Ax \leq b^u, \tag{5.5}$$

where $b^l = (\beta^l(\emptyset), \ldots, \beta^l(N))^T$ is the vector of known lower bounds and $b^u = (\beta^u(\emptyset), \ldots, \beta^u(N))^T$ the vector of known upper bounds. Note that we always have that $\beta^l(\emptyset) = \beta^u(\emptyset) = 1$. Assuming the synopses is up-to-date and was not built on a sample but the complete data, this system of inequalities is consistent and solvable. Thus, we can find solutions for x, the vector of γ-selectivities.

5.4.3 Estimating Selectivities: The Optimization Problem

Suppose we are given sampling bounds x^l and x^u as stated in Inequality 5.3 and, in addition, synopses bounds b^l and b^u as stated in Inequality 5.5. In this section, we show how to constrain an optimization problem to these bounds for the purpose of selectivity estimation.

For the objective function, we adopt the maximum entropy principle [58], since we consider it reasonable to assume independence in absence of knowledge. The entropy is maximized by the most uniform admissible solution. In vector

form, the entropy function is given by $-x^T \log(x)$ and can be maximized by minimizing its negative form. Note that in principle, every convex objective function allows one to find a global optimum.

Then, using the negated entropy function as objective function and the bounds from sampling and synopses as constraints, we formulate the constrained optimization problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & x^T \log(x) \\ \text{subject to} \quad & b^l \leq Ax \leq b^u, \\ & x^l \leq x \leq x^u. \end{aligned} \tag{5.6}$$

The solution vector x serves as an estimate for all γ-selectivities. By applying Equation 5.1 to all $X \subseteq N$, we obtain estimates for all β-selectivities.

Note that the optimization problem is subject to two types of constraints: (1) bounds on variables, often referred to as *box constraints*, and (2) linear inequality constraints. Furthermore, note that the objective function is strictly convex, which allows one to find the global minimum by searching for a local one. Strict convexity of a multidimensional function can be shown by proving that its Hessian matrix is positive definite.

Further note that, in rare cases, the solution space of Problem 5.6 is empty. One solution to this problem is to widen the bounds and try to solve the problem again. Another possibility is to find a solution that minimizes the constraint violation, e.g., in l_1-norm. We apply a solver that implements the latter, as discussed in Section 5.4.4.4.

Example: Suppose we want to estimate the selectivity of a conjunctive query that only contains one predicate p with index set $\{1\}$. In the preliminaries we observed that the selectivity of no predicate being applied $\beta(\emptyset)$ is always 1. However, assume for the sake of the graphical illustration of the optimization problem in Figure 5.4 that we only know $b^l = (\beta^l(\emptyset)) = (0.3)$ and $b^u = (\beta^u(\emptyset)) = (1)$ for the lower and upper bounds on β-selectivities, respectively. In addition, suppose that sampling, as described in Section 5.4.1, gives us $x^l = (\gamma^l(0), \gamma^l(1))^T = (0.1, 0.05)^T$ for the lower bounds and $x^u = (\gamma^u(0), \gamma^u(1))^T = (0.6, 0.7)^T$ for the upper bounds on γ-selectivities, where $\gamma(0)$ denotes the selectivity of $\neg p$ and $\gamma(1)$ denotes the selectivity of p. Then our knowledge induces the following optimization problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & x^T \log(x) \\ \text{subject to} \quad & (0.3) \leq \begin{pmatrix} 1 & 1 \end{pmatrix} x \leq (1), \\ & \begin{pmatrix} 0.1 \\ 0.05 \end{pmatrix} \leq x \leq \begin{pmatrix} 0.6 \\ 0.7 \end{pmatrix}, \end{aligned}$$

where $\begin{pmatrix} 1 & 1 \end{pmatrix}$ defines the 1×2 design matrix that corresponds to matrix A in optimization problem 5.6.

The optimal solution is $x = (0.367, 0.367)$ and is marked as the point of *Max Entropy* in Figure 5.4. Note that this point is no vertex. Furthermore, note

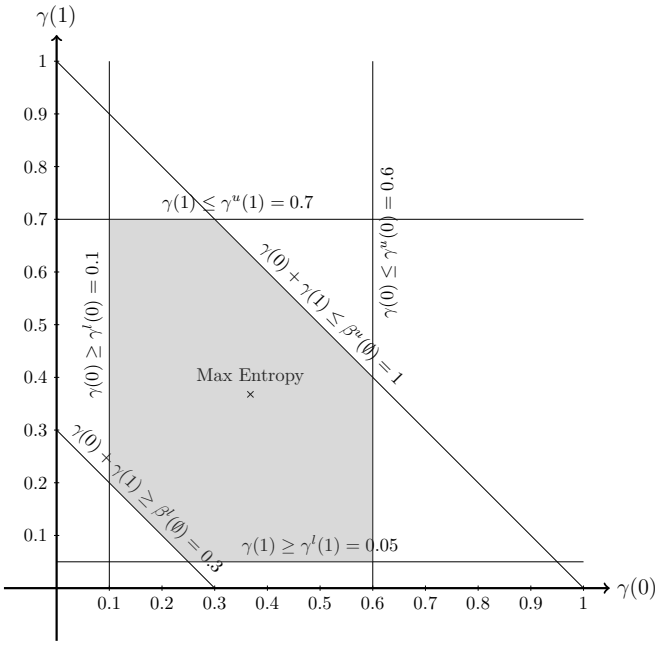


Figure 5.4: Graphical representation of an optimization problem example for a single predicate.

that the sum over all estimated γ-selectivities given in vector x does not add up to 1, since we did not set $\beta^l(\emptyset) = \beta^u(\emptyset) = 1$ in this example.

The final estimate for the selectivity of p, denoted by $\beta(1)$, is obtained by applying Equation 5.1, which yields $\beta(1) = \sum_{\{1\} \subseteq Y \subseteq N} \gamma(Y) = 0.367$.

5.4.4 Solving the Optimization Problem

In this section, we describe Mehrotra's predictor-corrector algorithm, which can be applied to Problem 5.6. Note that this section is complementary to the previous one: Those who are not interested in how to solve optimization problems may skip this section. We thank Oliver Kolb, former professor for mathematical optimization at University of Mannheim, for his assistance in the writing of this section.

Mehrotra's algorithm is well-established and implemented in optimization libraries such as IPOPT [87], which we apply. The algorithm belongs to the class of interior point methods, which find an optimal solution by following a path through the feasible region.

5.4.4.1 Rewrite

We rewrite the constraints in Problem 5.6 to simplify the discussion of Mehrotra's predictor-corrector algorithm. Note that no rewrite is required to apply IPOPT. After the rewrite of the constraints, the optimization problem is subject to greater-equals inequality constraints only. To this end, we leave $Ax \geq b^l$ unchanged and write $Ax \leq b^u \iff -Ax \geq -b^u$. For the variable constraints, we make use of the $2^n \times 2^n$ identity matrix I and rewrite $x \geq x^l$ to $Ix \geq x^l$ and $x \leq x^u$ to $-Ix \geq -x^u$.

Then, the rewritten problem is given by

$$\begin{aligned} &\text{minimize} && x^T \log(x) \\ &\text{subject to} && \begin{bmatrix} A \\ -A \\ I \\ -I \end{bmatrix} x \geq \begin{bmatrix} b^l \\ -b^u \\ x^l \\ -x^u \end{bmatrix}. \end{aligned} \tag{5.7}$$

We write $Mx \geq c$ as the short version of the constraints and denote by m the number of rows in M or c.

5.4.4.2 Optimality Conditions

An optimal solution to optimization problem 5.7 satisfies the Karush-Kuhn-Tucker conditions. In our case we have

$$\log(x) + e - M^T\lambda = 0 \tag{5.8a}$$
$$Mx - y - c = 0 \tag{5.8b}$$
$$y_i\lambda_i = 0 \qquad i = 1, 2, \ldots, m \tag{5.8c}$$
$$y, \lambda \geq 0 \tag{5.8d}$$

where 5.8a states that the gradient with respect to x of the Lagrangian for Problem 5.7 $\nabla_x \mathcal{L}(x, y, \lambda)$ must be zero. $\log(x) + e$ with $e = (1, 1, \ldots, 1)^T$ is the gradient of the objective function $x^T \log(x)$. Condition 5.8b states that the constraints in Problem 5.7 must hold. Here, $y \in R^{|G|+|G|+2^n+2^n}$ is a slack vector that compensates for the inequalities $Mx - c \geq 0$. The conditions 5.8c state that either (1) constraint i is *active*, meaning its slack variable y_i is zero and constraint i effectively imposes an equality constraint at this point, or (2) constraint i is inactive at this point, then its Lagrange multiplier λ_i must be zero. Hence, an equivalent way of writing conditions 5.8c is $(Mx - c)_i\lambda_i = 0$ for $i = 1, 2, \ldots, m$.

Since our objective function is strictly convex and all our constraints are linear, the aforementioned necessary conditions are also sufficient.

Mehrotra's predictor-corrector algorithm finds a solution that satisfies the optimality conditions in 5.8 in an iterative process. In each iteration, a new iterate is computed as

$$(x, y, \lambda) + \alpha(\Delta x, \Delta y, \Delta\lambda),$$

where (x, y, λ) is the current iterate, $(\Delta x, \Delta y, \Delta\lambda)$ is the search direction and α is the step size.

Conceptually speaking, while iterating, the search direction handles optimality conditions 5.8a-5.8c, while the step size selection provides an α that respects condition 5.8d and the *sufficient decrease* condition, which will be discussed later.

5.4.4.3 Search Direction

When determining the search direction we do two things. (1) We ignore 5.8d. (2) We do not force $y_i\lambda_i$ to be zero in 5.8c, but to be a pre-defined fraction $\sigma \in [0, 1]$ of the average value of the pairwise products in y and λ, i.e., $\mu := 1/m \sum_{i=1}^{m} y_i\lambda_i = y^T\lambda/m$. A choice of $\sigma > 0$ tends to allow for larger step sizes, since y and λ are biased towards positivity. The system we obtain for the search direction is

$$F(x, y, \lambda) := \begin{bmatrix} \log(x) + e - M^T\lambda \\ Mx - y - c \\ Y\Lambda e - \sigma\mu e \end{bmatrix} = 0, \tag{5.9}$$

where $Y := \text{diag}(y_1, y_2, \ldots, y_m)$ and $\Lambda := \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m)$.

Those solutions to Equation 5.9 that additionally satisfy $y, \lambda > 0$ represent the so-called *central path* that leads to an optimal solution as $\sigma\mu$ approaches zero, since the optimality conditions, stated in 5.8, are then satisfied.

We then formulate the following linear approximation $\hat{F}$ for F to predict a search direction

$$\hat{F}(x + \Delta x^p, y + \Delta y^p, \lambda + \Delta\lambda^p) = F(x, y, \lambda) + J(x, y, \lambda) \begin{bmatrix} \Delta x^p \\ \Delta y^p \\ \Delta\lambda^p \end{bmatrix},$$

where $J(x, y, \lambda)$ denotes the Jacobian of F.

Equations 5.9 tell us to find a root of F, and hence, we set $\hat{F}(x + \Delta x^p, y + \Delta y^p, \lambda + \Delta\lambda^p) = 0$. Computing $-F(x, y, \lambda)$ and plugging in the values for F and J, we get

$$\begin{bmatrix} L' & 0 & -M^T \\ M & -I & 0 \\ 0 & \Lambda & Y \end{bmatrix} \begin{bmatrix} \Delta x^p \\ \Delta y^p \\ \Delta\lambda^p \end{bmatrix} = \begin{bmatrix} -L + M^T\lambda \\ -Mx + y + c \\ -Y\Lambda e + \sigma\mu e \end{bmatrix}, \tag{5.10}$$

where $L := \log(x) + e$ and $L' := \frac{\partial L}{\partial x_i} = \text{diag}(1/x)$ denotes the Hessian of the objective function. Solving this system for $(\Delta x^p, \Delta y^p, \Delta\lambda^p)^T$ is called the predictor step. Its result can be used as a search direction. Note that for $\sigma = 0$ the search direction is the same as in Newton's method in optimization when solving 5.8a - 5.8c.

However, additionally performing a so-called correction step, defined by the following system, tends to reduce the number of iterations until convergence [70,

Chapt. 14.2, 16.6]

$$\begin{bmatrix} L' & 0 & -M^T \\ M & -I & 0 \\ 0 & \Lambda & Y \end{bmatrix} \begin{bmatrix} \Delta x^c \\ \Delta y^c \\ \Delta \lambda^c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\Delta X^p \Delta S^p e \end{bmatrix}, \tag{5.11}$$

where we solve for $(\Delta x^c, \Delta y^c, \Delta \lambda^c)^T$. Finally, the search direction for the current iteration becomes $(\Delta x^p, \Delta y^p, \Delta \lambda^p) + (\Delta x^c, \Delta y^c, \Delta \lambda^c)$.

5.4.4.4 Step Size

In each iteration, a step size α is selected such that the by now ignored condition 5.8d, stating $y, \lambda \geq 0$, is satisfied. The maximum step size we consider is $\alpha^{max} = \max\{\alpha \in (0,1] : (y,\lambda) + \alpha(\Delta y, \Delta\lambda) \geq \tau(y,\lambda)\}$, which satisfies the condition $(y,\lambda) + \alpha(\Delta y, \Delta\lambda) \geq 0$ with a buffer controlled by $\tau \in (0,1)$. In addition, a step size α must lead to a sufficient decrease of a so-called merit function. A merit function combines the two goals, reducing the objective function and satisfying the constraints, in one function. For our problem, a valid merit function is

$$\phi(x,y) = x^T log(x) - v||Mx - y - c||, \tag{5.12}$$

where v is a penalty parameter and can be chosen to be the largest Lagrange multiplier λ_i in λ, but many other choices exist [70, Chapt. 15.4, 18.3], and $||\cdot||$ can be chosen to be the l_1-norm. With regard to solvability of Problem 5.6, the merit function ϕ gives interesting insights. For a solvable problem, the second term of ϕ eventually vanishes, and the merit function coincides with the objective function. While given an unsolvable problem, as ϕ is decreased, we find a solution that minimizes the constraint violation.

Then, to find an $\alpha \in (0, \alpha^{max}]$ that provides a sufficient decrease of the merit function ϕ, we perform a backtracking line search, where we start with $\alpha = \alpha^{max}$ and iteratively decrease α until

$$\phi(x + \alpha\Delta x, y + \alpha\Delta y) \leq \phi(x,y) + \eta\alpha D_{\Delta x, \Delta y}\phi(x,y), \tag{5.13}$$

where $\eta \in (0,1)$ and $D_{\Delta x, \Delta y}\phi(x,y)$ denotes the directional derivative of $\phi(x,y)$ in the direction Δx and Δy, see [70, Chapt. A.2] for details.

5.4.4.5 Starting Point

A starting point has to satisfy only the positivity constraints $x, y, \lambda > 0$. In particular, it is not required to lay in the feasible region. However, the choice of the starting point impacts how fast the algorithm converges. Various heuristics exist [70, Chapt. 14.2, 16.6]. In our case, we choose $x_0 = (x^l + x^u)/2$ and leave the initialization of y and λ to IPOPT [87].

5.4.4.6 Exit Condition and Codification

CSE($b^l, b^u, x^l, x^u, \epsilon, maxDuration$)

```
 1  let Mx ≥ c be the constraints as written in Eq. 5.7
 2  let y be a slack vector
 3  Choose (x^(0), y^(0), λ^(0)) > 0
 4  k = 0
 5  maxTimePoint = TIME() + maxDuration
 6  repeat
 7      (x, y, λ) = (x^(k), y^(k), λ^(k))
 8      μ = y^T * λ/m
 9      Choose σ ∈ [0, 1]
10      Solve eq. 5.10 to obtain (Δx^p, Δy^p, Δλ^p)
11      Solve eq. 5.11 to obtain (Δx^c, Δy^c, Δλ^c)
12      (Δx, Δy, Δλ) = (Δx^p, Δy^p, Δλ^p)
                       +(Δx^c, Δy^c, Δλ^c)
13      Choose τ ∈ (0, 1)
14      α = max{α ∈ (0, 1] : (y, λ)+α(Δy, Δλ) ≥ τ(y, λ)}
15      while Eq. 5.13 not satisfied
16          α = α/2
17      (x^(k+1), y^(k+1), λ^(k+1)) = (x, y, λ)+α(Δx, Δy, Δλ)
18      k = k + 1
19  until ||(x^(k), y^(k), λ^(k))−(x^(k−1), y^(k−1), λ^(k−1))|| < ε
        or TIME() > maxTimePoint
20  return Cx_k
```

Ideally, we iterate until the optimality conditions, stated in 5.8, are satisfied. However, a practical convergence criterion is to terminate when the distance between consecutive iterates $||(x^{(k)}, y^{(k)}, \lambda^{(k)}) - (x^{(k-1)}, y^{(k-1)}, \lambda^{(k-1)})||$ is smaller than some small value ϵ, since we cannot expect to make significant progress beyond this point. Here, $||\cdot||$ denotes l_2-norm. In many applications, though, it is critical to obtain a fast estimate. We account for that by a maximum time span. Of course, time constraints provide no guarantees with respect to optimality.

Algorithm CSE shows the complete pseudo code. The parameters are synopses bounds b^l and b^u, sampling bounds x^l and x^u, as well as arguments to test the exit condition as described above. In line 1, the system $Mx \geq c$ is formulated as in Equation 5.7. Then, a slack vector y is introduced. In line 5, the latest time point for another iteration is determined. Then, in each iteration, μ and σ are computed to determine how much optimality condition 5.8c is relaxed. A simple strategy is to always choose $\sigma = 0$. For a description of adaptive choices, as used in IPOPT, see [70, Chapt. 14.2, 19.3]. Next, predictor and corrector steps are performed by solving Equation 5.10 and 5.11, respectively, to obtain a search direction. To determine a step size α, we first find the maximum step size α that preserves the positivity condition 5.8d with some specified buffer $\tau \in (0, 1)$, e.g., $\tau = 0.005$. Then, we iteratively halve α until the

sufficient decrease condition, given by Inequality 5.13, is satisfied. The current iterate plus a step of length α in the search direction gives the new iterate. After the last iteration, the vector of all estimated β-selectivities Cx_k is returned, where C denotes the complete design matrix introduced in Section 5.2.3.

5.5 Experimental Evaluation

We evaluate our approach (*CSE*) and compare it to the accuracy and run time of several existing estimation techniques that we have seen in the related work sections. To this end, we consider the sophisticated methods by Yu et al. (*HASE*) [95] and Markl et al. (*MaxEntropy*) [58]. In addition, we include Microsoft's exponential back-off estimator (*MsExpBackOff*) [9] as an up-to-date industry approach as well as simple random sampling (*Sampling*), i.e., the number of qualifying samples divided by the sample size, as a sampling-only estimator in our evaluation. Lastly, in the first part of our evaluation, we consider the estimates obtained by applying the independence assumption (*Ind. Ass.*). The following table shows for each estimator the type of information it processes:

Estimator	Sampling	Synopses
CSE	✓	✓
HASE	✓	✓
MaxEntropy		✓
MsExpBackOff		✓
Sampling	✓	
Ind. Ass.		✓

Note that HASE, MsExpBackOff and Ind. Ass. all process only one-dimensional synopses, i.e., single column statistics. Furthermore, note that MaxEntropy requires adjustment steps when processing multi-dimensional synopses as described in Section 5.3. For further details, recall the description of each model given in the related work section. If nothing else is mentioned, we model a data management system that provides estimators with a 1% sample of the data, as in related work [26, Chapt. 2], and one-dimensional histograms as well as two-dimensional histograms that capture correlations present in the data.

We use two real-world data sets in our experiments. (1) The *forest cover type* (forest) data set [56], which is popular in the machine learning community and contains more than 580k entries with 55 attributes. (2) The second data set represents joined data from the *daily global historical climatology network* (weather) [60, 61]. It comprises daily observations of climate records and contains about 3.4M entries with 7 attributes.

In our experiments, we consider different test scenarios that are defined by a data set and a number of predicates. For each test scenario, we run 10'000 conjunctive queries as in

SELECT $*$ **FROM** *dataset*
WHERE p_1 **and** ... **and** p_n

where each p_i represents a range predicate over an attribute A with constants c_1, c_2

$$A \textbf{ between } c_1 \textbf{ and } c_2$$

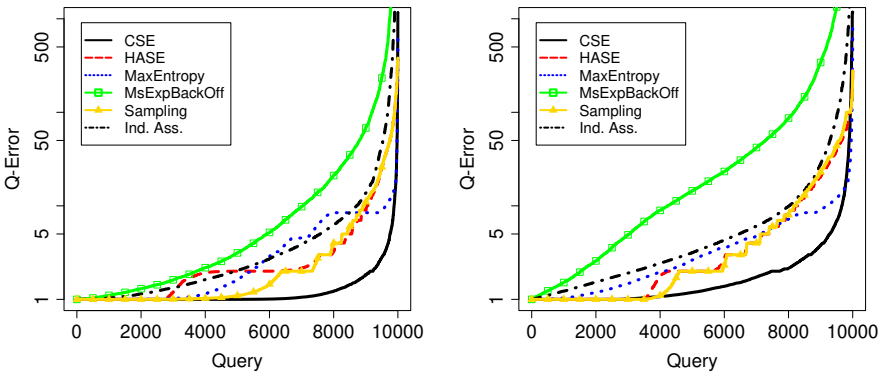
The range predicates p_i are generated by drawing a random unused attribute A and choosing two random values from A's domain. The smaller value is used for c_1 and the larger value is used for c_2. In case A's domain has only two values, e.g., 0 and 1, we set $c_1 = c_2$. This effectively creates a point predicate.

5.5.1 Accuracy

In this subsection, we look at the accuracy of estimates. The error metric used to measure the deviation between a selectivity estimate $\hat{s}$ and the true selectivity s is the q-error, which we have introduced in Section 3.2.1.

Notice that the q-error is undefined in case either the estimated or the true selectivity is zero. We configured all queries to return a non-empty result, hence, we do not have to worry about the true selectivity being zero. In addition, we programmed all estimators to estimate that at least one entry qualifies. Therefore, we do not have to worry about the estimated selectivity being zero. In a query optimization context, this makes sense to prevent faulty prunings in query plans.

5.5.1.1 The Idealistic Case: Synopses Without Approximation Errors



(a) 3 predicates **(b)** 5 predicates

Figure 5.5: Sorted q-errors for the weather data set given one and two-dimensional synopses without approximation errors and a 1% sample.

We first consider the idealistic case where histograms yield perfectly accurate selectivities. Note that in reality, no database system has access to such histograms. We include this case to pick up the scenario that was considered in the related works on MaxEntropy [58] and HASE [95], where approximation errors were not considered.

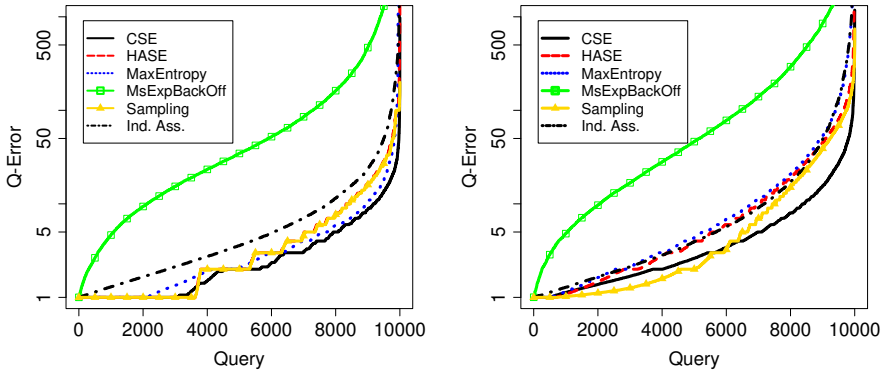
In [58] queries with 3 predicates find special attention. Figure 5.5a shows the q-errors sorted in ascending order for each estimator for queries with 3 predicates over the weather data set. Note that the longer a curve remains flat, the better the corresponding estimator. We make the following observations in Figure 5.5a. (1) CSE and HASE tend to be the best-performing models, supporting the idea of combining synopses and sampling. (2) HASE mostly resembles Sampling. (3) For some queries MaxEntropy outperforms Sampling, while for others, it is the other way around. In particular, as we will see later Sampling has good accuracy when enough samples qualify. The accuracy of MaxEntropy is good when the true selectivity is indeed near the point of maximum entropy subject to its constraints. (4) The difference between CSE and MaxEntropy or Sampling, respectively, illustrates the benefit of combining synopses and sampling as we propose it. (5) Microsoft's exponential back-off estimator is less accurate than the other estimators.

The largest queries considered in [95] contain 5 predicates. Figure 5.5b shows the accuracy of each estimator for queries of this size for the weather data set. Observe that (1) the curves of all estimators are shifted to the upper left as the number of predicates is increased from 3 to 5. This shift captures how selectivity estimation gets harder in the number of predicates in the conjunctive query. (2) The accuracy of HASE aligns with the accuracy of Sampling. (3) MaxEntropy worsens less than HASE or Sampling. (4) Microsoft's exponential back-off estimator performs significantly worse than all other estimators. Statically assuming a certain degree of correlation does not seem to be the key to success.

5.5.1.2 The Realistic Case: Synopses With Approximation Errors

In this subsection, we consider the realistic case where synopses structures, such as histograms, yield selectivities with approximation errors. We model q-optimal histograms [63], as can be found in SAP HANA [5], that guarantee a user-specified maximum multiplicative error for estimates, for which we choose a value of 2. Therefore, for each β-selectivity we want to provide to an estimator, we take the true β-selectivity, multiply it with a uniformly distributed multiplicative error in the range $[0.5, 2]$, and provide the product to the estimator.

Under this scenario, MaxEntropy requires adjustments of the provided selectivities as discussed in Section 5.3. We compute adjustments that are optimal under l_q, as described in [65]. CSE instead sets the lower bounds to $0.5 \times \beta(X)$ and the upper bounds to $2 \times \beta(X)$ for each provided selectivity $\beta(X)$ and is guaranteed to yield a feasible optimization problem. HASE, MsExpBackOff



(a) 7 predicates, weather data set **(b)** 7 predicates, forest data set

Figure 5.6: Sorted q-errors given one- and two-dimensional synopses with approximation errors and a 1% sample.

and Sampling operate as before, since they do not process multi-dimensional synopses.

Figure 5.6 shows the sorted q-errors for queries with 7 predicates for both the weather and the forest data set. Taking a closer look, we make the following observations. (1) Depending on the data set, synopses-based estimators like MaxEntropy or sampling-based estimators yield more accurate estimates, cf. Figure 5.6a, where MaxEntropy outperforms Sampling, and Figure 5.6b, where Sampling outperforms MaxEntropy. (2) Generally, CSE does not perform (significantly) worse than the best of MaxEntropy and Sampling. This makes it a robust estimator. In addition, in most cases CSE clearly produces the best estimates. (3) HASE performs no better than Sampling. (4) MsExpBackOff only considers the four most selective one-dimensional predicates, hence, it ignores three given selectivities. As a result, MsExpBackOff lags far behind the other estimators for both data sets.

5.5.1.3 Bias of Estimator

In this section we empirically analyze the bias of CSE. For this purpose we rely on the p-error, which functions like a signed version of the q-error. Recall, from Section 3.2.1, that the p-error is defined as $\text{p-error}(x, e) = \frac{e-x}{\min(e,x)}$ for a value x and an estimate e. Note that throughout this section, CSE uses a sample of size 1.000 tuples.

Figure 5.7 shows the p-errors for the forest data set for several estimators where the synopses that were provided for the estimators do not contain errors. As usual, the bottom and top of the box are the first and third quartiles, and

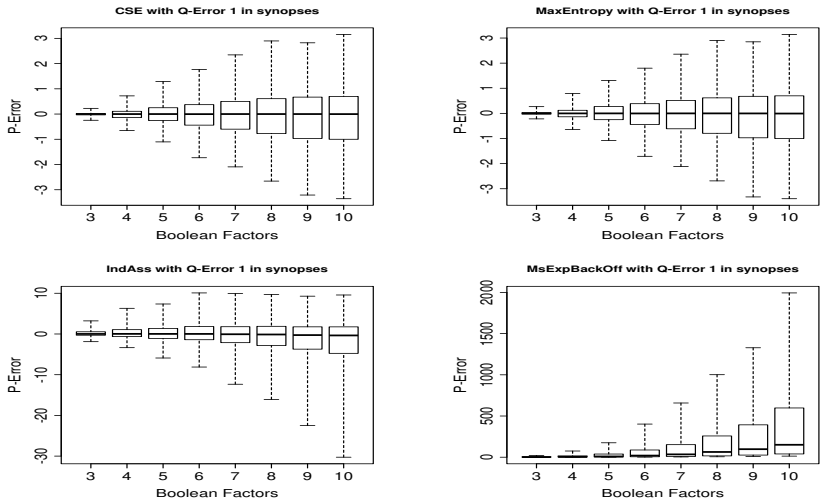


Figure 5.7: P-errors for the forest data set given perfect synopses.

the band in between represents the median. The lower and upper end of the whiskers represent the 1% and 99% percentile, respectively.

In the lower part of the picture, we see the independence assumption and Microsoft's exponential back-off estimator. While Independence assumption is median-unbiased, the factor by which it underestimates the cardinality can be much larger than the factor by which it overestimates the cardinality. Hence, Microsoft proposed the exponential back-off to correct for those severe underestimates. However, their estimator is biased towards overestimation and in some cases severely overestimates the result cardinality.

Looking at the plots for MaxEntropy and CSE, we observe that they seem to serve as unbiased estimators in this setting and underestimates are not significantly worse than overestimates. Hence, we conclude that 2-dimensional synopses without approximation errors suffice for the forest data set to produce unbiased estimates when assuming independence subject to the known synopses.

Note that we set the width of the synopses bounds of CSE to be the approximation error in the synopses. Hence, if synopses without approximation errors are given, the lower and upper synopses bounds of CSE are equal.

The picture for MaxEntropy and CSE changes when we consider synopses with approximation errors, cf. Figure 5.8. The errors, introduced as described in section 5.5.1.2, bias MaxEntropy and CSE towards underestimation. However, less severe in the case of CSE; note the difference in the scales for the plots of CSE and MaxEntropy.

Of course, dependencies are data set specific. Hence, in Figure 5.9, we look at the plots for the weather data set. Independence assumption is now rather biased towards overestimation. Even knowledge about 2-dimensional

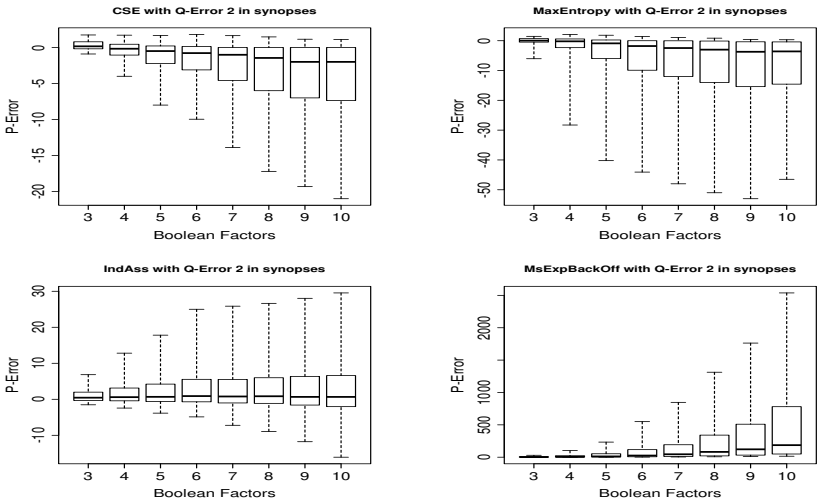


Figure 5.8: P-errors for the forest data set where synopses have approximation errors.

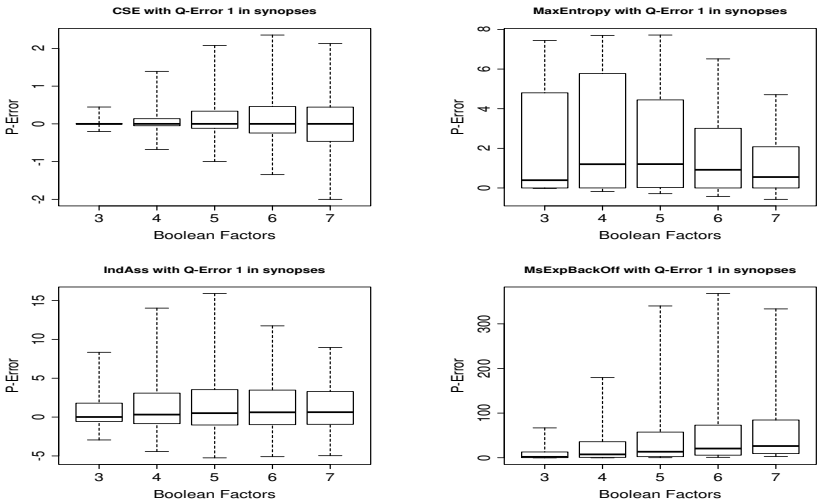


Figure 5.9: P-errors for the weather data set given perfect synopses.

dependencies does not completely change this pattern, as can be depicted from the plot for MaxEntropy. As before, Microsoft's exponential back-off estimator tends to overestimate result cardinalities, less severe, however, than in the forest data set. As before, CSE produces unbiased estimates.

As in the forest data set, synopses with approximation errors increase the underestimates for CSE and MaxEntropy rather than they increase the overestimates, cf. Figure 5.10.

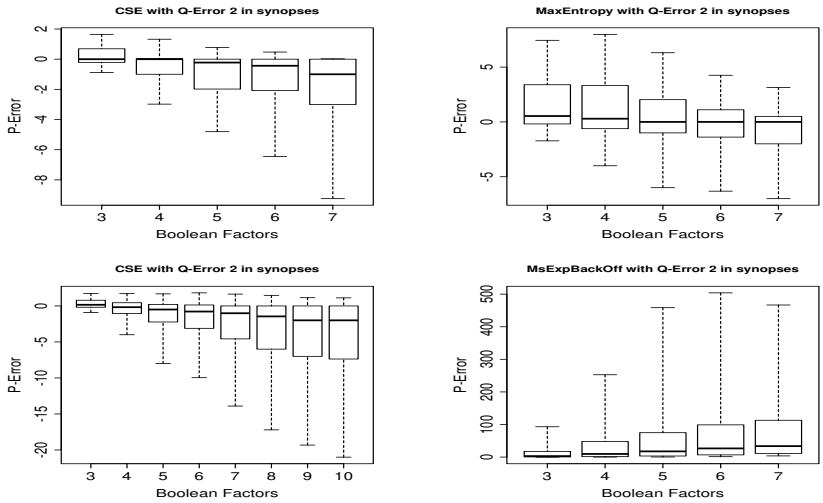


Figure 5.10: P-errors for the weather data set where synopses have approximation errors.

The errors of CSE strongly depend on the width of the synopses bounds, cf. Figure 5.11. In the title of the plots we refer to the width of the synopses bounds as *hist. width*. When the bounds are set to have a width of four (factor of two away from the given estimate in both directions) while the estimates do not contain errors, then the estimates still look as in the case where the synopses have approximation errors. Hence, the synopses bounds of CSE should be kept as tight as possible.

The last row in Figure 5.11 shows the reverse case where the width of the synopses bounds is 1, while the estimates contain approximation errors. Obviously, this setup yields to catastrophic estimates since, in case of overestimation, the true value is overestimated by a large factor.

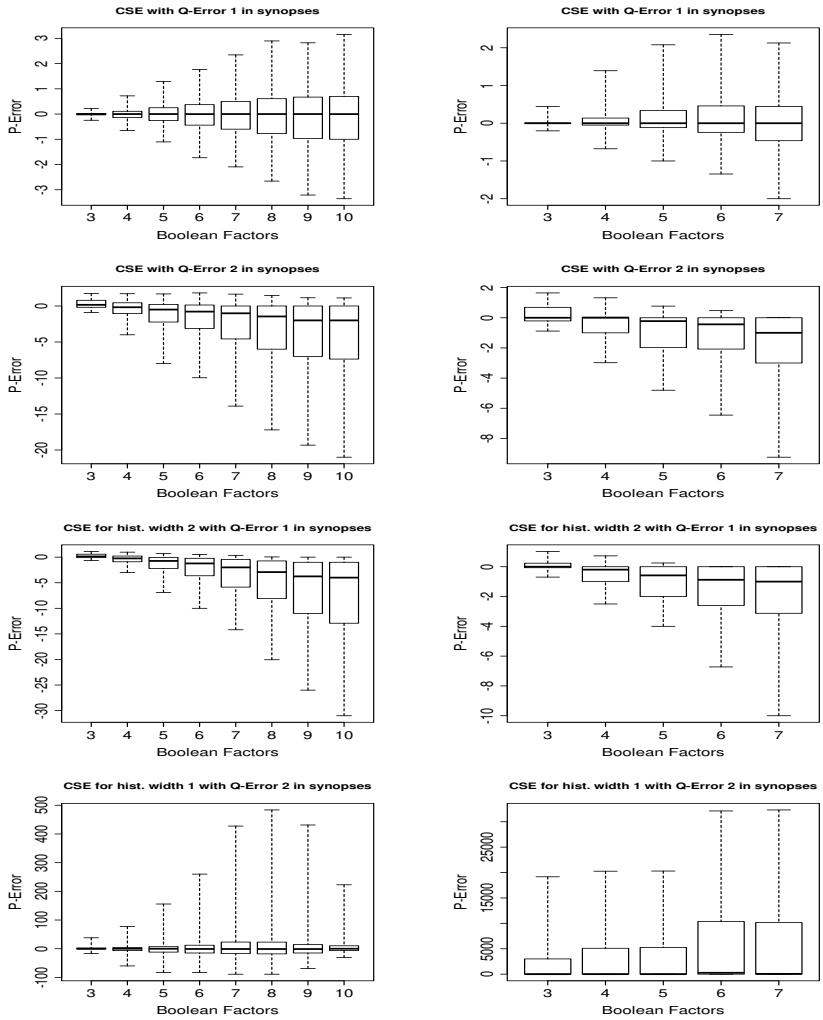
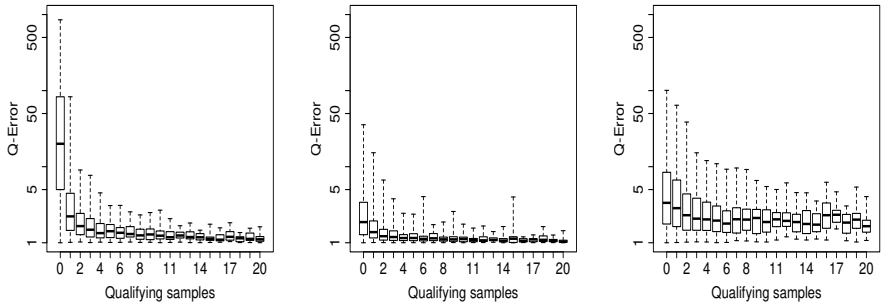


Figure 5.11: Impact of width of synopses bounds on p-error distribution of CSE in the forest data set.

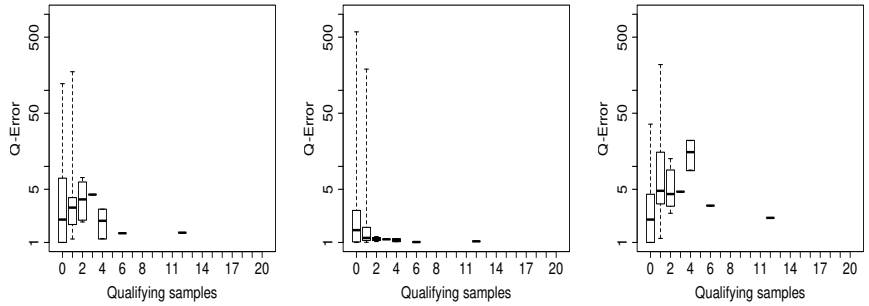
5.5.1.4 Common Aspects in Sampling and CSE

We now further investigate the impact of applying sampling, in addition to exploiting synopses, as we do it. Figure 5.12a shows a boxplot representation of the errors in the number of qualifying sample tuples for the forest data set. As usual, the bottom and top of the box are the first and third quartiles and the band in between represents the median. The lower and upper end of the whiskers



(a) Sampling **(b)** CSE, accurate synopses **(c)** CSE, inacc. synopses

Figure 5.12: Q-errors in the number of qualifying samples for 7 predicate queries and a sample size of 1.000 for the forest data set.



(a) Sampling **(b)** CSE, accurate synopses **(c)** CSE, inacc. synopses

Figure 5.13: Q-errors in the number of qualifying samples for 7 predicate queries and a sample size of 1.000 for the weather data set.

represent the 1% and 99% percentile, respectively. The last box aggregates the errors of all queries with more than 19 qualifying samples.

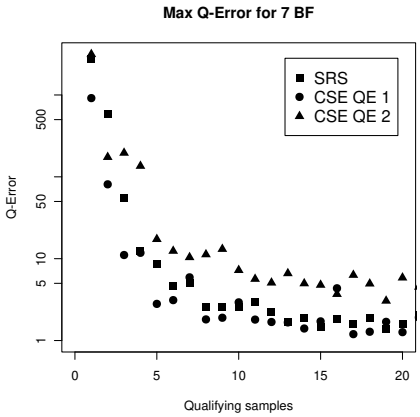
Observe how the errors of Sampling in Figure 5.12a decrease as more samples qualify, i.e., the estimates become more accurate. A practical way to see why the errors decrease is to observe how the confidence intervals close in in the number of qualifying samples. Since our approach incorporates sampling, it roughly follows this desirable trend, however, starting from a much lower error-level for few qualifying samples, cf. the y-axis-scales in Figures 5.12a with the ones 5.12b for CSE given accurate synopses and 5.12c for CSE given synopses with approximation errors (*inacc. synopses*). This is a competitive edge our approach gains over approaches that do not incorporate sampling. For comparison, Figure

5.13 shows the plots for the weather data set. Note that almost all samples had no qualifying tuple for the queries of the weather data set. When the boxplot for the q-errors is missing for a specific number of qualifying sample tuples, then there was no such sample.

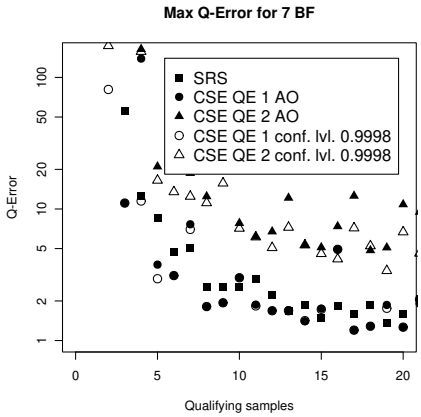
One might get the impression that sampling is the method of choice in almost all cases, since the errors become acceptable quickly in the number of qualifying samples. This is a misperception, since the case where very few samples qualify are the dominant ones. For instance, in the experiments conducted to produce the graphs in Figure 5.12, in more than 7,000 out of 10,000 queries with 7 simple predicates the sample had zero qualifying tuples. In the case of the weather data set, this is even more severe: 9968 out of 10,000 queries had zero qualifying tuples. Consequently, only 32 times there was a sample with qualifying tuples.

Figure 5.14 shows only the largest q-errors for simple random sampling, CSE with accurate synopses (QE 1), and CSE with synopses that contains approximation error (QE 2) for both the forest and the weather data set.

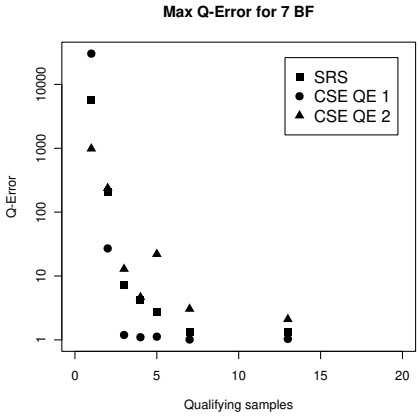
For comparison with a different technique to obtain sampling bounds, the plots in the right column (Figures 5.14b and 5.14d) show the maximum errors of CSE when using alpha/omega bounds, as described in [64], and Wilson Score intervals with a confidence level $1 - \alpha = 0.9998$. Note that in most cases the choice between alpha/omega and Wilson Score is irrelevant since the errors are located close to each other or coincide (when no difference can be observed), irrespective of whether the synopses is accurate or contains approximation errors. Taking a closer look we note that CSE using Wilson Score tends to produce more accurate estimates since we can identify cases where $\circ$ is closer to the x-axis than $\bullet$ and cases where Δ is closer the x-axis than $\blacktriangle$.



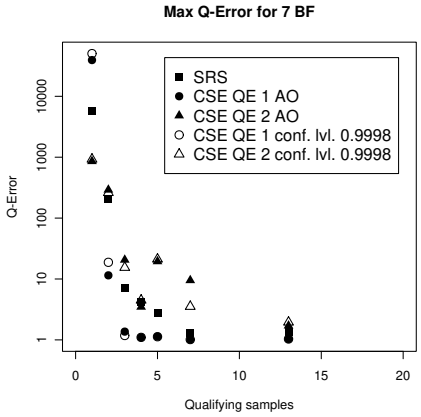
(a) forest data set. Simple Random Sampling and CSE with both synopses with and without approximation errors.



(b) forest data set, CSE with Alpha Omega bounds and $1 - \alpha = 0.9998$ conf. lvl intervals.



(c) weather data set. Simple Random Sampling and CSE with both synopses with and without approximation errors.



(d) weather data set, CSE with Alpha Omega bounds and $1 - \alpha = 0.9998$ conf. lvl intervals.

Figure 5.14: Max Q-Error in the number of qualifying tuples for 7 boolean factors

5.5.1.5 q-θ-acceptability

In this section we look at the q-θ-acceptability of various estimators. This measure puts estimates in two categories: *acceptable* and *unacceptable*. q-θ-acceptability is given if

- an estimate and the true value differ from each other by a factor of at most q
- or if both the estimate and the true value have a value of at most θ.

See [63] for details.

In this evaluation, we consider $q \in \{2, 4\}$ and $\theta \in \{100, 1000\}$. Tables 5.2, 5.3, and 5.4 illustrate the q-θ-acceptability for various estimators for the weather data set and the forest data set. In the headline of each table, *estimator* specifies the cardinality estimator, *synQe* denotes the q-errors in the provided synopses as described in Section 5.5.1.2, *noBf* denotes the number of simple predicates per conjunctive query, and *total* the number of queries evaluated with this estimator. Each column *qXtY* denotes the number of queries that are X-Y-acceptable, in the sense of the above definition of q-θ-acceptability.

In addition to the estimators we have already seen in the previous sections, we include SmplMe, that is, we derive the most independent solution subject to sampling bounds. This is the result we obtain when we run CSE without synopses bounds. We included SmplMe to demonstrate that it is a bad approach, as can be seen by looking at the q-θ-acceptability rates of SmplMe for any number of boolean factors. In the absence of synopses it is better to rely on classic sampling estimators, like SmplSrs.

Independence assumption and Microsoft exponential back-off have their limitations, cf. for instance Table 5.3 column q2t100 where, even without errors in the provided synopses, for 5 boolean factors, only 65% of the IndAss estimates and only 16% of the MsExpBackOff estimates are acceptable.

The advantage of CSE is its robustness in comparison to MaxEntropy and SimplSrs. For instance in Table 5.3, column q2t100, noBf=5, and synQe=2, we have that only around 6100 out of 10.000 MaxEntropy estimates are acceptable but around 7100 of the CSE estimates are. In Table 5.2, column q2t100, noBf=3 SmplSrs produces around 8200 acceptable estimates, whereas CSE produces around 9600 acceptable estimates in case of no errors in the synopses, i.e., synQe=1, and still 8800 acceptable estimates when we have synQe=2.

estimator	synQe	noBf	total	q2t100	q2t1000	q4t100	q4t1000
SmplMe	-	3	10000	1295	1295	1852	1852
SmplSrs	-	3	10000	8260	9035	8500	9275
CSE	1	3	10000	9628	9820	9760	9874
Hase	1	3	10000	7620	8437	8649	9435
MaxEntropy	1	3	10000	9745	9908	9877	9960
MsExpBackOff	1	3	10000	6514	7557	7348	8245
IndAss	1	3	10000	8279	8826	8906	9268
CSE	2	3	10000	8880	9282	9622	9808
Hase	2	3	10000	7723	8542	8453	9242
MaxEntropy	2	3	10000	8712	9205	9340	9611
MsExpBackOff	2	3	10000	5932	7093	6845	7852
IndAss	2	3	10000	7517	8136	8549	8962
SmplMe	-	5	10000	231	231	332	332
SmplSrs	-	5	10000	9261	9762	9329	9830
CSE	1	5	10000	9534	9820	9678	9855
Hase	1	5	10000	9172	9677	9344	9848
MaxEntropy	1	5	10000	9676	9936	9824	9966
MsExpBackOff	1	5	10000	5744	7965	5999	8142
IndAss	1	5	10000	8619	9321	8922	9459
CSE	2	5	10000	9282	9728	9624	9901
Hase	2	5	10000	9157	9663	9285	9790
MaxEntropy	2	5	10000	9080	9591	9348	9730
MsExpBackOff	2	5	10000	5394	7712	5667	7892
IndAss	2	5	10000	8186	9049	8544	9255
SmplMe	-	7	10000	8	8	18	18
SmplSrs	-	7	10000	9848	9967	9859	9978
CSE	1	7	10000	9722	9868	9762	9870
Hase	1	7	10000	9849	9968	9860	9979
MaxEntropy	1	7	10000	9940	10000	9979	10000
MsExpBackOff	1	7	10000	5102	8729	5197	8737
IndAss	1	7	10000	9495	9911	9587	9923
CSE	2	7	10000	9853	9969	9904	9981
Hase	2	7	10000	9846	9967	9860	9980
MaxEntropy	2	7	10000	9785	9915	9815	9920
MsExpBackOff	2	7	10000	4649	8458	4719	8465
IndAss	2	7	10000	9191	9824	9301	9842

Table 5.2: q-θ-acceptability of various estimators for the weather data set

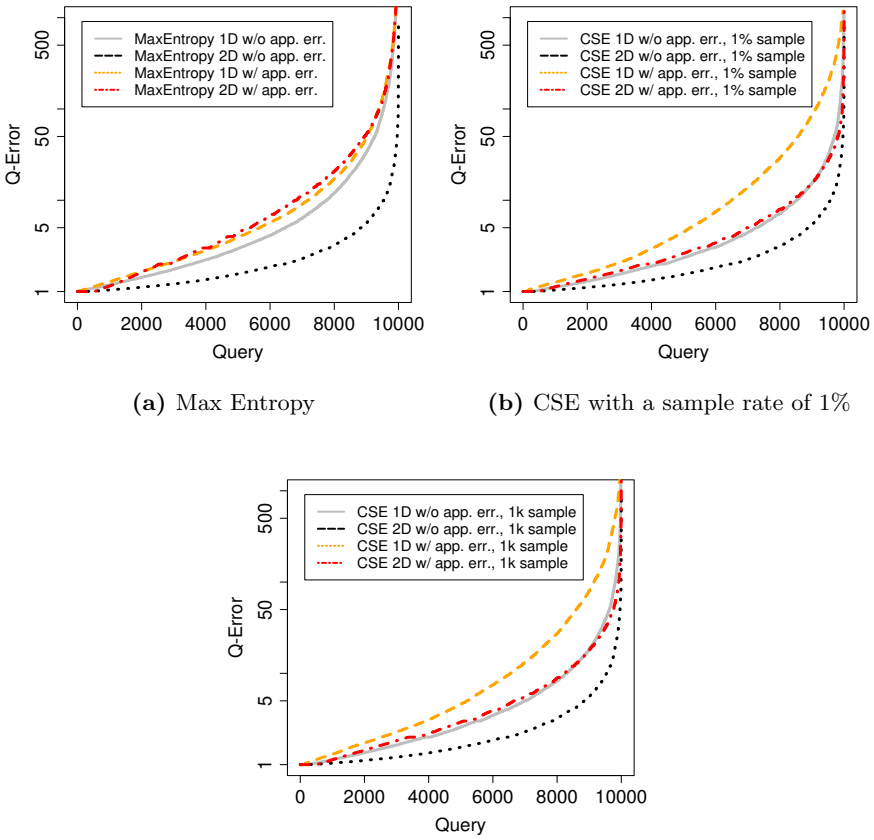
estimator	synQe	noBf	total	q2t100	q2t1000	q4t100	q4t1000
SmplMe	-	3	10000	3219	3219	4896	4896
SmplSrs	-	3	10000	8175	9364	8792	9833
CSE	1	3	10000	9752	9929	9910	9972
Hase	1	3	10000	5732	6973	8800	9744
MaxEntropy	1	3	10000	9787	9953	9937	9987
MsExpBackOff	1	3	10000	4738	5790	6857	7608
IndAss	1	3	10000	7504	8537	8766	9353
CSE	2	3	10000	7870	8561	9409	9692
Hase	2	3	10000	6135	7395	8226	9246
MaxEntropy	2	3	10000	7278	8136	8804	9259
MsExpBackOff	2	3	10000	3696	4702	6033	6783
IndAss	2	3	10000	5683	6745	8061	8697
SmplMe	-	5	10000	1787	1787	2650	2650
SmplSrs	-	5	10000	7331	9373	7978	9830
CSE	1	5	10000	9267	9922	9776	9971
Hase	1	5	10000	6265	8380	7995	9736
MaxEntropy	1	5	10000	9261	9922	9776	9976
MsExpBackOff	1	5	10000	1604	3301	3113	4610
IndAss	1	5	10000	6578	8572	8128	9327
CSE	2	5	10000	7147	8854	8759	9715
Hase	2	5	10000	6029	8079	7357	9152
MaxEntropy	2	5	10000	6171	8159	7599	8982
MsExpBackOff	2	5	10000	1383	2892	2591	3951
IndAss	2	5	10000	5109	7175	7027	8406

Table 5.3: q-θ-acceptability of various estimators for the forest data set: 3 and 5 boolean factors.

estimator	synQe	noBf	total	q2t100	q2t1000	q4t100	q4t1000
SmplMe	-	7	10000	1209	1293	2225	2307
SmplSrs	-	7	10000	7544	9581	8069	9886
CSE	1	7	10000	9070	9910	9685	9969
Hase	1	7	10000	7213	9253	8158	9850
MaxEntropy	1	7	10000	9053	9907	9682	9971
MsExpBackOff	1	7	10000	787	2669	1296	3073
IndAss	1	7	10000	6817	8990	8148	9518
CSE	2	7	10000	7215	9118	8541	9701
Hase	2	7	10000	6809	8757	7539	9292
MaxEntropy	2	7	10000	6739	8963	7710	9413
MsExpBackOff	2	7	10000	704	2383	1133	2713
IndAss	2	7	10000	5571	7987	6971	8661
SmplMe	-	10	10000	1137	9561	2416	9777
SmplSrs	-	10	10000	8069	9689	8448	9897
CSE	1	10	10000	9242	9966	9734	9992
Hase	1	10	10000	8078	9693	8567	9911
MaxEntropy	1	10	10000	9224	9966	9729	9994
MsExpBackOff	1	10	10000	426	2148	505	2196
IndAss	1	10	10000	7710	9553	8508	9747
CSE	2	10	10000	8025	9610	8683	9819
Hase	2	10	10000	7862	9384	8183	9580
MaxEntropy	2	10	10000	7931	9600	8348	9740
MsExpBackOff	2	10	10000	385	1862	458	1910
IndAss	2	10	10000	6627	8956	7478	9213

Table 5.4: q-θ-acceptability of various estimators for the forest data set: 7 and 10 boolean factors.

5.5.2 Dimensionality of the Given Synopses and Sample Size



(a) Max Entropy

(b) CSE with a sample rate of 1%

(c) CSE with a sample of size 1000

Figure 5.15: Sorted q-errors for queries with 7 predicates for the forest data set.

Multi-attribute statistics are well-established in the research community. However, many database systems still maintain only single attribute statistics. In this subsection, we show how the accuracy of MaxEntropy and CSE changes as we provide one- or two-dimensional synopses. Ideally, more information results in more accurate estimates. Indeed, we observe that for each estimator in each test scenario the estimates improve as we go from one-dimensional synopses to two-dimensional synopses if the selectivities provided by the synopses structures have no approximation errors. As discussed earlier, though, this is

an unrealistic assumption. In the realistic case, where synopses are subject to approximation errors, we observe in some test scenarios that the estimates become worse as more approximated selectivities were provided. Figure 5.15a compares the sorted q-errors of MaxEntropy for one-dimensional (1D) and one- and two-dimensional (2D) synopses. Note how the estimates improve in the idealistic case of synopses without approximation errors (w/o app. err.) but how they worsen in the case of synopses with approximation errors (w/ app. err.). This means that an estimator might perform better using less than all available information. However, this effect is data-dependent. For the weather data set, we do not observe this effect: more synopses always result in better estimates in all test scenarios for the weather data set.

Figure 5.15b shows the same graphs for CSE. This time in both cases, synopses with or without approximation errors, the estimates improve as we go from one-dimensional to one- and two-dimensional synopses. Since the same holds for the weather data set and different numbers of predicates, we conclude that CSE is robust in the sense that additional synopses have no negative impact on the estimates. Note that it is irrelevant how CSE 2D inacc performs in comparison to CSE 1D acc, since CSE 1D acc represents a hypothetical case. Figure 5.15c illustrates that the graphs look similar even if the sample used in CSE is only of size 1.000. Compared to the 1% sample, a sample of size 1.000 is more than a factor of 5 smaller for the forest data set. We observed that the results with a sample of size 1.000 are similar in many cases. From an industry point of view, that is good news since many database systems use such small samples.

Finally, note that CSE is designed for multi-dimensional synopses in combination with a sample, and we advise its application primarily in that context. In principle, CSE can be even applied to a sample only. However, it is not advisable to do so since the most independent solution subject to sampling bounds is a terribly wrong estimate!

5.5.3 Information Gain

This short section shows that 1-dimensional and 2-dimensional synopses do not suffice to capture all dependencies in a data set. To this end, we show that the forest data set serves as a counter example. In particular, if 1-dimensional and 2-dimensional synopses would capture all dependencies, then higher-dimensional synopses would not improve the estimates of an estimator that fully exploits all available information. MaxEntropy is such an estimator.

Figure 5.16 shows the q-errors for estimates produced by MaxEntropy given synopses of different dimensions for the forest data set. In the plot, xD means we provide MaxEntropy with x-dimensional synopses and y-dimensional synopses, for all $y < x$. For instance, 3D means we provide MaxEntropy with 3-dimensional synopses as well as 2-dimensional and 1-dimensional synopses. Note that the synopses are free of approximation errors.

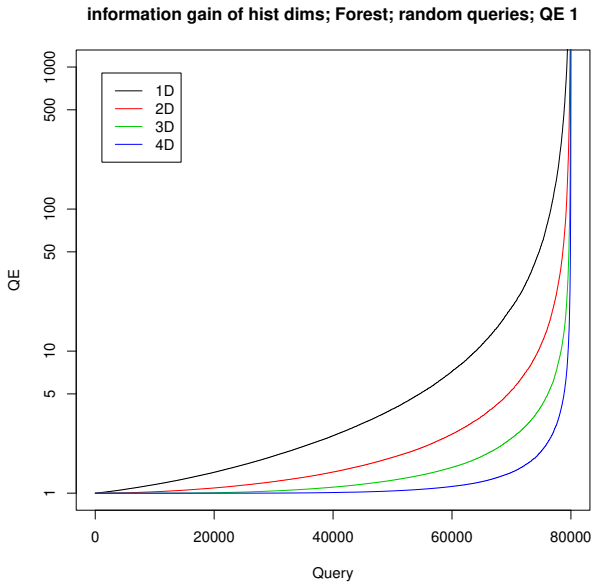


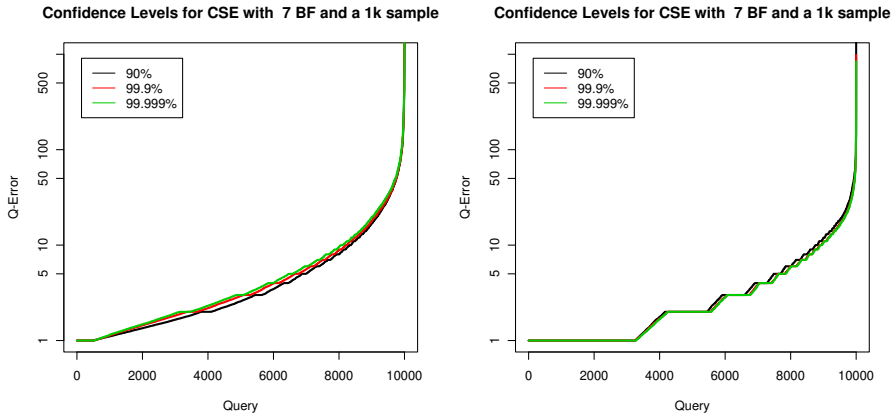
Figure 5.16: Information gained by histograms of additional dimensionality

By looking at Figure 5.16, we conclude that higher-dimensional synopses helps to improve estimates. Hence, dependencies that are not captured by 1-dimensional and 2-dimensional synopses exist.

5.5.4 Confidence Level of Sampling Bounds

To produce a selectivity estimate as described in Section 5.4.3, CSE uses sampling bounds. In Section 5.4.1 we showed how to compute the Wilson score confidence interval as one possibility to derive sampling bounds. A parameter for confidence intervals is the error rate α, which determines the confidence level $1 - \alpha$. We use $\alpha = 10^-3$ to produce intervals with a confidence level of 99.9% in all experiments in the other sections of the evaluation. In this section, we show that this value of α is a reasonable choice and, in particular, that CSE is not very sensitive to the value chosen for α.

In this section, we consider $1 - \alpha \in \{90\%, 99.9\%, 99.999\%\}$. Figure 5.17 shows the plot for a sample size of 1.000 tuples for the forest data set and weather data set where all queries contained 7 boolean factors. (1) Note that the differences are rather small. (2) In Figure 5.17a a confidence level of just 90% yields the best estimates followed by a confidence level of 99.9% an then 99.999%. However, in 5.17b the order is reversed. Hence, $1 - \alpha = 99.9\%$ represents a middle way.



(a) forest data set **(b)** weather data set

Figure 5.17: Confidence Levels for 1k sample. Synopses with approximation errors.

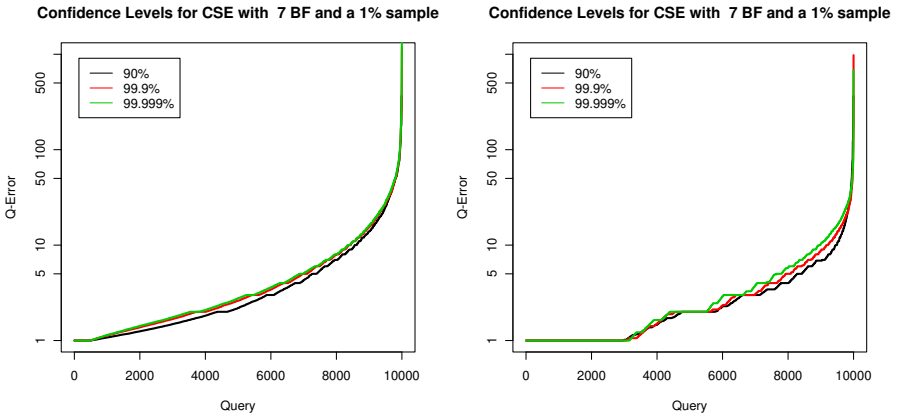
Figure 5.18 shows the same plots as Figure 5.17, but for a sample size of 1%. Recall that for both data sets, forest and weather, a sample size of 1% results in samples of a size that is much larger than 1.000 tuples. Note that for both data sets $1 - \alpha = 90\%$ gives the best estimates. However, the optimization problem underlying CSE finds an optimal solution *only* in 99.91% of the cases. Whereas, for $1 - \alpha = 99.9\%$, it finds the optimal solution in 99.98% of the cases. Hence, $1 - \alpha = 99.9\%$ results in a more robust estimator.

Robustness can greatly impact the accuracy of estimates for specific queries. We now investigate queries where CSE did not find an optimal solution for $1 - \alpha \in \{90\%, 99.9\%, 99.999\%\}$. Note that until now our plots showed the sorted q-errors of each estimator *individually*. In the following, we present tables that show for specific queries the q-errors obtained for CSE with $1 - \alpha \in \{90\%, 99.9\%, 99.999\%\}$. We include queries with different boolean factors, indicated by *noBf*. The column *Exit* shows the exit status of the solver for the optimization problem:

- ✓ denotes *Solve_Succeeded* or *Solved_To_Acceptable_Level*
- ⏲ denotes *User_Requested_Stop*
- The dots ... denote *Maximum_Iterations_Exceeded*
- ↯ denotes *Infeasible_Problem_Detected* (never observed)

Note that the solver terminates if $iter > 10 \wedge lTime > 1s \vee iter > 3000$ where the first term of the disjunction refers to ⏲ and the second refers to the dots ... .

Prominent cases where a larger confidence level $1 - \alpha$ increases accuracy can be found in the forest data set where the sampling bounds were derived from



(a) forest data set **(b)** weather data set

Figure 5.18: Confidence Levels for 1% sample. Synopses with approximation errors.

a sample of size 1.000 where the q-error for CSE with $1 - \alpha = 90\%$ was larger than 2000, while for CSE with $1 - \alpha = 99.9\%$ it was lower than 2. Similarly, in the weather data set where the sampling bounds were derived from a sample of size 1.000, we can find queries where CSE with $1 - \alpha = 90\%$ produced an estimate that was off by a factor 12.000, while CSE with $1 - \alpha = 90\%$ estimated the true selectivity.

Hence, there exist cases where finding the optimal solution with respect to the optimization problem is important. We conclude that $1 - \alpha$ should be chosen to be small as long as the chance is small that CSE does not find an optimal solution.

queryIdx	noBf	QE,90%	Exit	QE,99.9%	Exit	QE,99.999%	Exit
6744	3	2086.70	...	1.94	✓	1.94	✓
57	3	462.00	...	4.32	✓	1.72	✓
725	3	234.19	...	1.80	✓	1.74	✓
5840	3	94.87	...	5.27	✓	3.45	✓
2785	3	74.00	...	3.09	✓	2.80	✓
378	3	72.00	...	3.65	✓	3.65	✓
6447	3	70.26	...	2.40	✓	2.40	✓
3072	3	62.00	...	1.09	✓	1.09	✓
2724	3	58.00	...	1.29	✓	1.29	✓
6109	3	37.00	...	2.65	✓	2.65	✓
3021	3	35.00	...	1.19	✓	1.19	✓
351	3	30.78	...	1.21	✓	1.84	✓
2734	3	19.96	...	6.15	...	3.54	...
4388	3	14.00	...	14.00	✓	14.00	✓
1394	3	12.00	...	1.77	✓	1.55	✓
4353	3	9.24	...	5.36	✓	4.87	✓
5901	3	5.06	...	2.48	✓	2.41	✓
2907	3	5.01	...	1.18	✓	1.16	✓
3811	3	5.00	...	1.85	✓	1.85	✓
3009	3	4.75	...	1.74	✓	1.74	✓
6539	3	4.06	...	2.34	✓	2.23	✓
1411	3	3.78	...	3.10	✓	2.91	✓
5046	3	3.46	...	1.07	✓	1.07	✓
4156	3	3.41	...	1.50	✓	1.02	✓
6715	3	3.30	...	1.07	✓	2.04	✓
3272	3	3.00	...	4.01	✓	2.48	✓
5458	3	2.91	...	1.28	✓	1.10	✓
3403	3	2.65	...	1.27	✓	1.20	✓
6801	3	2.55	...	1.30	✓	1.14	✓
3318	3	2.43	...	1.07	✓	1.36	✓
2810	3	2.22	...	1.46	✓	2.54	✓
3717	3	2.01	...	1.13	✓	1.17	✓
5247	3	2.00	...	1.19	✓	1.19	✓
2219	3	2.00	...	3.69	✓	3.73	✓
7	3	1.89	...	1.25	✓	1.28	✓
3409	3	1.88	...	3.00	✓	2.86	✓
6119	3	1.64	...	1.16	✓	1.38	✓
582	3	1.64	...	1.09	✓	1.33	✓
5895	3	1.36	...	1.95	✓	1.96	✓
1565	3	1.31	...	1.96	✓	2.53	✓
2849	3	1.28	...	1.32	✓	1.30	✓
6649	3	1.24	...	3.10	✓	2.98	✓
807	3	1.17	...	1.71	✓	1.80	✓
3144	3	1.17	...	1.26	✓	1.25	✓
6110	3	1.10	...	1.05	✓	1.17	✓
5710	3	1.06	...	1.08	✓	1.14	✓
6650	3	1.05	...	2.01	✓	2.01	✓
3118	3	1.04	...	1.32	✓	1.37	✓
4138	3	1.00	...	1.94	✓	2.85	✓
5722	3	1.00	...	1.00	...	1.00	...

Table 5.5: Forest data set and 1k sample. Synopses with approximation errors.

queryIdx	noBf	QE,90%	Exit	QE,99.9%	Exit	QE,99.999%	Exit
4061	3	90.01	...	1.49	✓	1.49	✓
16334	4	70.02	...	5.00	✓	5.00	✓
6975	3	57.00	...	1.25	✓	1.65	✓
975	3	40.00	...	9.78	✓	9.78	✓
49635	7	18.00	⏲	18.00	✓	18.00	✓
8865	3	10.83	...	1.03	✓	1.03	✓
3055	3	9.00	...	1.27	✓	1.27	✓
5947	3	8.00	...	2.31	✓	2.31	✓
9660	3	8.00	...	1.48	✓	1.48	✓
3072	3	5.81	...	12.60	✓	12.72	✓
15874	4	5.71	...	3.93	✓	5.04	✓
7033	3	5.49	...	1.51	✓	1.31	✓
10215	4	4.92	...	2.57	✓	2.57	✓
725	3	4.73	...	1.83	✓	1.12	✓
3302	3	4.00	...	3.71	✓	3.71	✓
1563	3	3.44	...	1.06	✓	1.02	✓
2734	3	3.44	...	1.06	✓	1.64	✓
7264	3	3.26	...	5.07	✓	5.24	✓
18471	4	3.17	...	1.52	✓	1.00	✓
47684	7	3.00	⏲	3.00	✓	3.00	✓
5093	3	2.51	...	2.29	✓	2.39	✓
2810	3	2.37	...	1.09	✓	1.79	✓
9578	3	2.14	...	1.06	✓	1.43	✓
3792	3	2.13	...	1.05	✓	1.05	✓
9686	3	2.03	...	1.28	✓	2.09	✓
49920	7	2.00	⏲	2.00	⏲	2.00	✓
50330	8	2.00	⏲	2.00	✓	2.00	✓
4737	3	1.96	...	1.16	✓	1.50	✓
2907	3	1.85	...	1.20	✓	1.19	✓
1583	3	1.72	...	1.21	✓	1.21	✓
6010	3	1.65	...	5.36	✓	9.18	✓
14375	4	1.56	...	1.19	✓	1.72	✓
9212	3	1.54	...	3.99	✓	6.00	✓
4291	3	1.53	...	1.17	✓	1.67	✓
5523	3	1.50	...	1.04	✓	1.04	✓
3933	3	1.42	...	1.13	✓	1.13	✓
5306	3	1.32	...	1.77	✓	2.16	✓
4921	3	1.24	...	1.51	✓	1.73	✓
9810	3	1.18	...	1.46	✓	1.69	✓
9991	3	1.08	...	1.10	✓	1.14	✓
7508	3	1.06	...	1.08	✓	1.18	✓
1801	3	1.06	...	1.09	✓	1.20	✓
6699	3	1.03	...	1.18	✓	1.30	✓
2509	3	1.00	...	1.00	✓	1.00	✓
8109	3	1.00	...	1.97	✓	1.97	✓
19394	4	1.00	...	1.00	✓	1.00	✓
20858	5	1.00	⏲	1.00	✓	1.00	✓
30441	6	1.00	⏲	1.00	✓	1.00	✓
49405	7	1.00	⏲	1.00	✓	1.00	✓
54879	8	1.00	⏲	1.00	⏲	1.00	⏲

Table 5.6: Forest data set and 1% sample. Synopses with approximation errors.

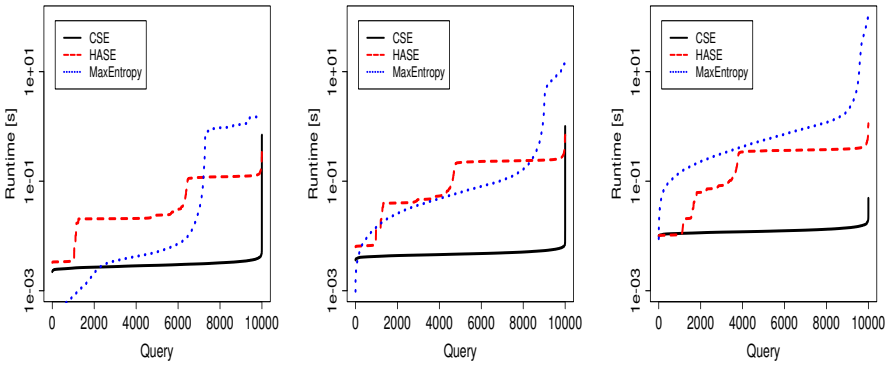
queryIdx	noBf	QE,90%	Exit	QE,99.9%	Exit	QE,99.999%	Exit
27	3	12486.80	...	3.63	✓	3.63	✓
39	3	12481.40	...	1.00	✓	1.00	✓
249	3	12481.40	...	1.00	✓	1.00	✓
472	3	12481.40	...	16.29	✓	16.29	✓
523	3	6240.69	...	3.20	✓	5.31	✓
597	3	656.91	...	1.37	✓	1.62	✓
545	3	124.81	...	13.80	✓	13.80	✓
132	3	124.15	...	2.18	✓	2.13	✓
282	3	71.69	...	1.12	✓	1.12	✓
103	3	52.00	...	1.84	✓	1.35	✓
125	3	37.00	...	37.00	✓	37.00	✓
176	3	29.85	...	9.20	...	5.30	...
308	3	28.00	...	1.59	✓	1.62	✓
210	3	13.27	...	4.09	...	2.35	...
390	3	12.48	...	2.58	✓	4.59	✓
89	3	12.00	...	2.02	✓	2.11	✓
444	3	12.00	...	2.59	✓	2.81	✓
220	3	11.37	...	3.50	✓	2.02	✓
525	3	10.38	...	3.20	✓	1.84	✓
171	3	7.02	...	2.16	✓	1.93	✓
576	3	6.67	...	1.60	✓	1.46	✓
366	3	6.36	...	2.50	✓	2.50	✓
106	3	6.00	...	1.78	✓	1.78	✓
226	3	5.00	...	5.00	...	5.00	...
143	3	4.64	...	2.70	✓	2.86	✓
40	3	3.00	...	1.07	✓	1.09	✓
65	3	3.00	...	5.95	✓	2.83	✓
221	3	3.00	...	1.38	✓	1.38	✓
461	3	2.47	...	1.95	✓	1.95	✓
246	3	2.13	...	1.02	✓	1.02	✓
2	3	2.00	...	11.56	✓	11.56	✓
86	3	2.00	...	2.00	✓	2.00	✓
204	3	2.00	...	1.45	✓	1.45	✓
222	3	2.00	...	315.17	✓	315.17	✓
372	3	2.00	...	24.53	✓	27.59	✓
463	3	2.00	...	2.33	✓	5.17	✓
497	3	2.00	...	1.35	✓	1.35	✓
287	3	1.44	...	2.24	✓	2.24	✓
601	3	1.32	...	1.80	✓	2.05	✓
573	3	1.21	...	3.06	✓	3.80	✓
205	3	1.00	...	1.00	✓	1.00	✓
225	3	1.00	...	4.66	✓	3.36	✓
236	3	1.00	...	1.00	✓	1.00	✓
248	3	1.00	...	1.59	✓	1.59	✓
340	3	1.00	...	1.00	✓	1.00	✓
361	3	1.00	...	8.43	✓	8.43	✓
465	3	1.00	...	1.63	✓	1.63	✓
539	3	1.00	...	1.84	✓	1.84	✓
543	3	1.00	...	1.00	✓	1.00	✓
574	3	1.00	...	1.00	✓	1.00	✓

Table 5.7: Weather data set and 1k sample. Synopses with approximation errors.

queryIdx	noBf	QE,90%	Exit	QE,99.9%	Exit	QE,99.999%	Exit
35	3	363.77	...	1.00	✓	1.00	✓
6234	3	360.17	...	2.14	✓	2.14	✓
6834	3	360.15	...	2.50	✓	2.49	✓
1368	3	180.09	...	1.22	✓	1.22	✓
3976	3	180.09	...	2.00	✓	2.00	✓
5690	3	94.60	...	1.00	✓	1.00	✓
7242	3	46.46	...	1.25	✓	1.25	✓
7128	3	45.16	...	1.85	✓	1.85	✓
6181	3	38.98	...	4.55	✓	4.53	✓
10407	4	37.72	...	6.34	✓	6.34	✓
10009	4	6.87	...	2.12	...	1.22	✓
10043	4	6.87	...	2.12	✓	1.22	✓
10063	4	6.87	...	2.12	✓	1.22	✓
10136	4	6.87	...	2.12	✓	1.22	✓
10241	4	6.87	...	2.12	✓	1.22	✓
10446	4	6.87	...	2.12	...	1.22	✓
10477	4	6.87	...	2.12	✓	1.22	✓
8224	3	4.73	...	1.83	✓	1.12	✓
7831	3	4.73	...	1.89	✓	1.89	✓
3051	3	4.26	...	1.65	✓	1.00	✓
7862	3	4.26	...	1.68	✓	1.00	✓
8432	3	3.63	...	2.84	✓	2.84	✓
8279	3	3.60	...	1.23	✓	1.23	✓
10018	4	3.44	...	1.06	✓	1.64	✓
10281	4	3.44	...	1.06	✓	1.64	✓
10714	4	3.44	...	1.06	✓	1.25	✓
4045	3	3.05	...	1.19	✓	1.13	✓
3634	3	3.04	...	1.18	✓	1.38	✓
10280	4	2.90	...	1.07	✓	1.05	✓
3758	3	2.82	...	1.18	✓	1.17	✓
3139	3	2.51	...	1.35	✓	1.07	✓
7232	3	2.29	...	1.42	✓	2.46	✓
10297	4	2.29	...	1.42	✓	2.46	✓
3394	3	2.13	...	1.21	✓	1.47	✓
7410	3	2.13	...	1.09	✓	1.13	✓
6019	3	2.00	...	1.26	✓	1.26	✓
666	3	1.86	...	1.18	✓	1.13	✓
1729	3	1.80	...	1.59	✓	1.59	✓
5097	3	1.80	...	2.25	✓	2.24	✓
6926	3	1.73	...	1.96	✓	1.96	✓
10488	4	1.72	...	1.89	✓	2.35	✓
10530	4	1.72	...	1.89	✓	2.26	✓
2307	3	1.42	...	1.44	✓	1.44	✓
731	3	1.18	...	1.04	✓	1.00	✓
8881	3	1.16	...	1.19	✓	1.48	✓
471	3	1.09	...	1.01	✓	1.08	✓
8135	3	1.07	...	1.02	✓	1.08	✓
10029	4	1.00	...	1.41	✓	1.00	✓
10166	4	1.00	...	3.62	✓	4.27	✓
10742	4	1.00	...	1.00	✓	1.32	✓

Table 5.8: Weather data set and 1% sample. Synopses with approximation errors.

5.5.5 Runtime



(a) 3 predicates **(b)** 5 predicates **(c)** 7 predicates

Figure 5.19: Sorted runtimes given one- and two-dimensional synopses with approximation errors.

We performed single-threaded runtime measurements on a machine with Intel Skylake i5-6500 CPU with a clock rate of 3.20GHz with 16GB RAM. The machine is operated by a 64-bit linux. We employ the IPOPT library [87] to solve the optimization problem underlying our approach.

We only look at the run times of CSE, HASE and MaxEntropy. Our implementation of HASE applies the algorithm based on Newton's method with a multiplicative distance function. Both the algorithm and the distance function are described in the paper [95]. By MaxEntropy we refer to the approach by Markl et al. that applies iterative scaling [58] to solve the underlying optimization problem. Note that Sampling and MsExpBackOff run orders of magnitude faster since they perform only simple arithmetics instead of solving optimization problems. Furthermore, we do not consider the cost of drawing a sample or extracting information from synopses structures; we are only interested in the time it takes to produce an estimate, given some information.

Figure 5.19 shows the run times of CSE, HASE and MaxEntropy for several problem sizes, i.e., various numbers of simple predicates. Note that we consider the realistic case where synopses are subject to approximation errors.

Looking at Figure 5.19a, we note that (1) MaxEntropy works best in about 20% of all queries containing three simple predicates. That is, the overhead to get the algorithm started is low. (2) Furthermore, the graph of MaxEntropy suggests that the runtime heavily depends on the known selectivities that define the constraints. (3) HASE and our approach CSE both seem to be much more independent of the values of the selectivities in the optimization problem. (4) HASE runs slower than our approach.

To see how the algorithms scale in the problem size, we analyze Figures 5.19a, 5.19b and 5.19c. Looking at the graphs of MaxEntropy, we observe that the runtime grows exponentially in the number of predicates which the authors stated themselves [58]. In their future work section, they suspect an algorithm based on Newton's method to be faster. Indeed, this has been shown to be true [36]. In addition, note that our approach solves the maximum entropy approach, as Markl et al. have stated it, when setting all lower and upper bounds on the variables to 0 and 1, respectively, and imposing equality constraints on the β-selectivities.

Hase and CSE both scale well. Looking closely, we observe that the relative difference between HASE and CSE shrinks, i.e., HASE has better asymptotic properties. Let us look at the changes in the underlying optimization problem as we increase the number of simple predicates form n to $n+2$: The number of variables in the optimization problem quadruples since the number of γ-selectivities grows from 2^n to $2^{n+2} = 4 \times 2^n$ for both HASE and CSE. Only in CSE, a box constraint is associated with each variable, cf. Problem 5.6. For both approaches, the number of constraints induced by one-dimensional synopses grows from n to $n+2$. Additionally, CSE has constraints due to two-dimensional synopses which grow by a factor of $4n+2$ from $\frac{n(n-1)}{2}$ to $\frac{(n+2)((n+2)-1)}{2} = (4n+2)\frac{n(n-1)}{2}$.

5.6 Summary

We proposed CSE, a novel approach to combine sampling with synopses for the purpose of estimating the selectivity of conjunctive queries. The results of our experiments suggest that CSE indeed leads to more accurate selectivity estimates. Using two real-world data sets and a large number of queries, we attempted to show the strengths and limitations of our own approach and various other state-of-the-art approaches. Depending on the patterns in the data set, a purely sampling-based estimator or a purely synopses-based estimator yields better selectivity estimates. Our approach, however, yields estimates that are at least as accurate as the estimates of the best competing estimator. This makes CSE a robust estimator, whose applicability does not depend on the data set.

Chapter 6

Sketches for Intersection Size Estimation

In this section, we discuss the sketches AKMV and HyperLogLog and how to use them for intersection size estimation. Both sketches, and the techniques for intersection size estimation, become relevant in later chapters of this thesis. Note that, unlike the other chapters, this chapter discusses and analyses only existing techniques.

6.1 AKMV Sketch

AKMV sketches play an important role in Section 7 as well as Section 8. Hence, this section introduces AKMV sketches and their predecessor KMV sketches. As part of the following discussion, we use relations R and T with attribute sets C and D, respectively. The k *minimum value sketch* (KMV) [15] is a synopsis that allows to estimate the number of distinct values (NODV) in a multiset. In our context, the multiset under consideration consists of the values in $R.C$. A KMV is constructed in a single pass over $R.C$ and requires only a constant amount of memory. The basic idea is simple: To construct a KMV, hash each entry in $R.C$ to $[0, 1]$ and keep track of the k smallest hashes, where k is some fixed number. We refer to the tracked hashes by the totally ordered set $\mathcal{H} := \{h_1 < h_2 < \cdots < h_k\}$. Then, to estimate the NODV in $R.C$, use h_k as an indicator. In particular, a large h_k indicates few distinct entries in $R.C$ and, vice versa, a small h_k indicates many distinct input entries in $R.C$. As a formula, the KMV estimate for the NODV is

$$\widehat{d}_{KMV} = \begin{cases} k/h_k & \text{, if } d > k \\ k & \text{, else} \end{cases} \tag{6.1}$$

where $d := |R.C|_d := |\pi^D_C(R)|$, i.e., the number of distinct values in $R.C$. As the formula suggests, for fewer than k distinct values, the estimate is the true value.

For simplicity, we use only one deterministic global hash function $H : \circ \to [0,1]$ that is capable of hashing any input to $[0,1]$. To simplify the exposition, we always assume the common case $d > k$. Unless explicitly stated otherwise, all KMVs/AKMVs share the same fixed parameter k.

The *augmented KMV* (AKMV) by Beyer et al. [18] is an extension of KMV. The main idea of AKMV is to augment the KMV by counters to track the multiplicity by which each hash is seen during construction. The counters enable one to estimate the NODV in a multiset that is the intersection, union, or difference of other multisets. For instance, three AKMVs can be used to estimate the NODV in the intersection of three multisets. In the following, we formally define AKMVs, introduce the multiset operations they support, and show how AKMVs are used for NODV estimation.

We formally define an AKMV for $R.C$ as $\mathcal{S}_{R.C} := (\mathcal{H}_{R.C}, \eta_{R.C})$, where $\mathcal{H}_{R.C}$ denotes the set of the k smallest hashes we know from KMV, and the function $\eta_{R.C} : [0,1] \to \mathbb{N}$ returns the tracked multiplicity of h if $h \in \mathcal{H}_{R.C}$. Otherwise, if $h \notin \mathcal{H}_{R.C}$, we define $\eta_{R.C}(h) = 0$.

AKMVs support the multiset operations union $\cup$, intersection $\cap$, and multiset difference $\setminus$. Let $\mathcal{S}_{R.C}, \mathcal{S}_{T.D}$ be two AKMVs. The result of $\mathcal{S}_{R.C} \circ \mathcal{S}_{T.D}$, where $\circ \in \{\cup, \cap, \setminus\}$, is a new AKMV $\mathcal{S}_{R.C \circ T.D}$. For brevity, let $E := R.C \circ T.D$. The set of hashes $\mathcal{H}_E$ in $\mathcal{S}_E$ is defined as the k smallest hashes in $\mathcal{H}_{R.C} \cup \mathcal{H}_{T.D}$ and each $h \in \mathcal{H}_E$ has multiplicity

$$\eta_E(h) = \begin{cases} \eta_{R.C}(h) + \eta_{T.D}(h) & \text{, if } \circ = \cup \\ \min(\eta_{R.C}(h), \eta_{T.D}(h)) & \text{, if } \circ = \cap \\ \max(\eta_{R.C}(h) - \eta_{T.D}(h), 0) & \text{, if } \circ = \setminus \end{cases} \tag{6.2}$$

Note that intersect and subtract operations can cause hashes with a multiplicity of zero. In case of intersection, this indicates that some entry was present only in one of $R.\mathcal{C}$ and $T.D$.

The appropriate NODV estimate for an arbitrary AKMV $\mathcal{S}$ must reflect the number of hashes with multiplicity greater zero, to which we refer as $p := |\{h_i \in \mathcal{H} \mid \eta(h_i) > 0\}|$. Equation 6.1 is not applicable for AKMVs, since p is not reflected. Instead, the formula to compute the NODV estimate $\widehat{d}_{AKMV}$ of $\mathcal{S}$ is

$$\widehat{d}_{AKMV} = \frac{p}{k} \cdot \frac{k-1}{\max(\mathcal{H})} \tag{6.3}$$

In the formula, the first fraction is the ratio of tuples with multiplicity greater zero. The second fraction is almost the KMV estimate from Equation 6.1 - Beyer et al. have shown that decrementing the numerator by 1 is necessary to make the estimator unbiased [18].

Example: Consider AKMV $\mathcal{S} = (\mathcal{H}, \eta) = (\{0.0002 < 0.003 < 0.008 < 0.015\}, (30, 50, 0, 40))$. Observe that $\max(\mathcal{H})$=0.015 and p=3 hashes have multiplicity greater zero. Hence, assuming k=4, the NODV estimate $\widehat{d}_{AKMV}$ of $\mathcal{S}$ is $3/4 \cdot (4-1)/0.015 = 150$.

6.2 HyperLogLog

This section first introduces the HyperLogLog sketch. Originally, HyperLogLog was designed for NODV estimation only. Then, we present three techniques to estimate intersection sizes using HyperLogLog sketches. Two of these techniques were invented only in the recent years.

6.2.1 HyperLogLog Sketch

Finding the NODV in a stream/data set in a single pass over the data is easy using memory linear in the NODV. For instance, using a hash set, one can find the *exact* NODV with memory $O(n)$ for n distinct values.

A HyperLogLog sketch (HLL) [32] allows to *estimate* the NODV in a single pass over the data with much less memory. The idea is simple: Suppose you are interested in the number of different people in your various WhatsApp group chats. Assume that the last few digits in a phone number are (uniformly) random. Then, since digits range from 0-9, the chance that a specific digit in a phone number has value 0 is 1/10. Hence, the chance that a phone number ends in 00 is 1/100. Put differently, one expects that you have group-chatted with around 100 different people if one phone number ends in 00 and no other phone number ends in more zeros.

HLL applies this logic to binary numbers. Equal distribution of the input numbers is achieved via hashing. To avoid heavy overestimates through a hash value that, by coincidence, ends in many zeros, HLL applies *stochastic averaging* [32]. That is, the first k bits of an b-bit hash value are used as an index in an array of 2^k counters. The remaining $b-k$ bits are used to count the number of trailing zeros. Each counter stores the maximal number of trailing zeros (plus one) among all hash values mapped to this counter. Finally, to estimate the NODV, the (harmonic) mean over the counters is computed. Since each counter only processed an average portion of $1/2^k$ of the input domain, the final estimate is the mean scaled up by a factor of 2^k (and multiplied by a constant).

The memory footprint of an HLL is very small. Since n distinct hash values can be represented with $\log_2(n)$ bits, and we can count the number of zeros in a hash value with $\log_2$ of the hash value's bits, the memory consumption of each counter is $\log_2(\log_2(n))$. Hence, with $m = 64$ counters as in [33], an HLL for a stream with up to 10^9 distinct values requires only $64 \cdot log_2(log_2(10^9)) \approx 0.3$kb.

It can be shown that the estimator is practically unbiased. The variance decreases proportionally in the number of counters. To be more precise, the standard error is $\frac{1.04}{\sqrt{m}}$ for an HLL of m counters.

Consider the following example to illustrate the memory consumption and estimation error. With $m = 64$ counters, we approximately determine the NODV in a stream with up to 10^9 distinct values with only $64 \cdot log_2(log_2(10^9)) \approx 0.3kb$. In real applications, one usually uses 1 byte per counter, so 64B in total [33]. According to Flajolet, due to the central limit theorem, estimation errors follow a normal distribution. Hence, we expect to overestimate in only 1% of the cases by more than an additive term of $2.326 \cdot \frac{1.04}{\sqrt{64}} = 0.3024$, where

2.326 is the 99th percentile of the normal distribution. By symmetry of the normal distribution, the same argument holds for underestimation.

6.2.2 HyperLogLog Intersection Size Estimation

To the best of our knowledge, the three established approaches are: (1) The inclusion-exclusion principle, which builds on basic set theory, (2) Ertl's approach, which is based on a Poisson model [29], and (3) the binomial mean lookup (BML) approach from Microsoft by Nazi et al. [68]. [68] do not explicitly mention intersection size but the *inclusion coefficient* $\frac{|M_1 \cap M_2|}{|M_1|}$, where M_1, M_2 are multisets. However, an intermediate result of their method is an estimate for the intersection size.

In the following discussion, we denote the two HLLs that are intersected by H^K and H^F, and use K and F as shorthand notations for $\pi_K(R)$ and $\pi_F(S)$:

6.2.2.1 Inclusion-Exclusion Principle

Asking for the number of unique values in some multiset or data column is equivalent to asking for the cardinality of a set filled with the same data. From basic set theory we know that

$$|K \cup F| = |K| + |F| - |K \cap F|,$$

and can simply solve for the intersection size

$$|K \cap F| = |K| + |F| - |K \cup F|.$$

To estimate $|K \cup F|$, we must merge the two HLL sketches H^K, H^F. Let H^K, H^F have m counters each and denote by $H[i]$ the value of the ith counter. Then, $H^{K \cup F}$ with counter values $H^{K \cup F}[i] = \max(H^K[i], H^F[i]), 0 \leq i < m$ is the result of merging H^K with H^F. The NODV estimate of $H^{K \cup F}$ is an estimate for the NODV in $K \cup F$. In fact, this estimate is also equal to the NODV estimate of an HLL built on $K \cup F$ [29].

It is possible that the HLL estimate for $|K \cup F|$ exceeds the estimate for $|K| + |F|$. In this case, as a sanity bound, one should return 0 or a small default minimum intersection size to prevent faulty prunings in query plans, e.g. 0.5.

Another notable effect of the inclusion-exclusion principle occurs for HLL sketches where the values of all counters in K's sketch are greater than the values of all counters in F's sketch. In this case, the estimates for $|K|$ and $|K \cup F|$ are equal. Therefore, the estimate for $|K \cap F|$ is $|F|$, i.e., the upper bound of the intersection size.

6.2.2.2 Binomial Mean Lookup

The key aspect of BML is a multivariate function f that maps $|K|$, $|F|$ and $|K \cap F|$ to the probability that $H^K[i] \leq H^F[i]$, i.e.,

$$f(|K|, |F|, |K \cap F|) \rightsquigarrow \Pr(H^K[i] \leq H^F[i]).$$

The definition of f is lengthy and involves different cases, it can be found in the paper [68]. Therefore, we use $\rightsquigarrow$ to describe the output.

Giving a frequentist argument, [68] define the probability $\Pr(H^K[i] \leq H^F[i])$ as the relative frequency

$$\Pr(H^K[i] \leq H^F[i]) := \frac{\sum_{j=0}^{m-1} \mathbb{1}_{H^K[j] \leq H^F[j]}}{m}, \tag{6.4}$$

where $\mathbb{1}$ denotes the indicator function. Since $\Pr(H^K[i] \leq H^F[i])$ is the same for all counters i, we simply write P.

In order to estimate the intersection size $|K \cap F|$, one must conceptually compute f^{-1}. Nazi et al. argue that f is monotonically increasing and, therefore, f^{-1} exists [68]. $|K|$ and $|F|$ are considered as constants since their values are approximately known from H^K and H^F, and, hence,

$$f^{-1}_{|K|,|F|}(P) \rightsquigarrow |K \cap F|.$$

However, [68] do not explicitly define the function f^{-1}. Instead, they use the function f and the observed probability P, defined as above in Eq. (6.4), to test different values for $x \in [0, \max(|K|, |F|)]$ until $f(|K|, |F|, x) = P$ is found. The matching x is the estimate for the intersection size. Since f is monotonically increasing [68], the authors propose bisection method/binary search to find the intersection size estimate in logarithmic time $O(\log(\max(|K|, |F|)))$. More efficient algorithms are Brent's method [19] and TOMS 748 [12].

6.2.2.3 Ertl's Approach

In his statistical model [29], Ertl assumes that the number of distinct values and the counter values in an HLL sketch are Poisson distributed. The Poisson distribution's single non-negative parameter λ expresses the rate at which new unique values are inserted into the sketch.

To estimate the intersection size of two HLL sketches H^K and H^F, Ertl assumes that pairwise distinct elements are inserted into H^K and H^F at rates λ_K and λ_F, respectively. In addition, further unique elements are inserted into both sketches at rates $\lambda_\cap$. That is, $\lambda_\cap$ defines the intersection size of H^K and H^F. Note that λ_K, λ_F, and $\lambda_\cap$ are unknown, and the goal is to approximate $\lambda_\cap$ in order to estimate the intersection size.

Ertl derives the joint probability mass function f that gives the probability that the counters in H^K equal the values in a vector C^K and that the counters in H^F equal the values in a vector C^F, denoted by $\Pr(H_i^K = c^K \wedge H_i^F = c^F)$:

$$f(\lambda_K, \lambda_F, \lambda_\cap, C^K, C^F) \rightarrow \prod_{i=1}^{n} \Pr(H_i^K = C_i^K \wedge H_i^F = C_i^F),$$

where $\Pr(H_i^K = c^K \wedge H_i^F = c^F)$ denotes the probability that counter i in H^K equals entry i in C^K. The Poisson assumption implies independence of HLL counters and, hence, probabilities can be multiplied. The definition of $\Pr(H_i^K = c^K \wedge H_i^F = c^F)$ can be found in the original paper [29].

For C^K and C^F, Ertl chooses the counter values observed in H^K and H^F. Then we must solve the optimization problem

$$\max_{\lambda_K,\lambda_F,\lambda_\cap} f(\lambda_K, \lambda_F, \lambda_\cap \mid C^K, C^F)$$

subject to the constraint that $\lambda_K, \lambda_F, \lambda_\cap > 0$.

After finding an optimal solution and the corresponding optimal parameter $\lambda^*_\cap$, the expected intersection size is computed. Since $\mathbb{E}[Poisson(\lambda^*_\cap)] = \lambda^*_\cap$, the estimated intersection size is simply $\lambda^*_\cap$.

Ertl points out that there is more than one optimal solution to the above optimization problem. Hence, the estimated intersection size depends on the solution we find. In Newton's method the optimal solution depends on the initial values for the parameters $\lambda_K, \lambda_F, \lambda_\cap$. Ertl suggests to use values derived based on the Inclusion Exclusion Principle, which we presented in 6.2.2.1. This often yields an initial solution that is already an optimal solution or close to one.

6.3 Evaluation

6.3.1 Hash Function

As we have seen, both AKMV and HLL sketches can be used for NODV estimation and, more importantly in the context of this thesis, intersection size estimation. For both sketches, their size and hash function are important parameters. For AKMV, size corresponds to the number of hash values stored. For HLL, size corresponds to the number of counters. Both use a hash function to hash the elements from the multiset they represent.

For the hash functions, we consider the functions listed below, where `x` is the input value to be hashed. `c1`, `c2`, `c3` denote constants that are chosen randomly but only once before the first call to any of the hash functions. By a**b we denote a^b. Recall that AKMV computes estimates based on normalized hash values in $[0, 1]$. This normalization is not captured by the below hash functions. To ensure AKMV operates correctly, either each computed hash must be normalized. This is the approach we take in our implementation. Or one operates on integer hash values and only normalizes to compute an estimate.

name	definition
murmur	`x ^= x >> 16; x *= 0x85ebca6b; x ^= x >> 13;` `x *= 0xc2b2ae35; x ^= x >> 16; return x;`
fibonacci	`b = 2654435769; a = b * (1.0 / 2**32 );` `z = a * x; return floor(n * (z - floor(z)));`
boncz	`return ((x >> 21) ^ (x >> 13) ^ (x >> 7) ^ (x));`
crc32	`return __builtin_ia32_crc32si(x, c1);`
poly1	`return x * c1 + c2;`
poly4	`return x**3 * c1 + x**2 * c2 + x * c3 + c4`

To analyze the accuracy of AKMV and HLL, we perform the following experiment, in which the hash function h and size m are parameters: For 100

sketches, all of the same type $X \in \{\text{HLL}, \text{AKMV}\}$, with size parameter m and hash function h, we incrementally insert 1 million values into each sketch. After each insert, we record the largest q-error among all 100 sketches for the estimate of the current NODV. Finally, we output the largest q-error over all inserts.

In Table 6.1 and Table 6.2, we report the results of the aforementioned experiment for different hash functions and sizes. As shown in the second column of the tables, we chose the size parameter such that, in the ith row, both sketches have the same memory footprint. For AKMV, all hash functions have a reasonable accuracy, as the number of values stored grows large. For HLL, fibonacci and boncz perform notably bad. We favor hash functions that perform reasonable well in both sketches. For the remainder of this section, we fix the hash function for AKMV to poly 1 and the hash function for HLL to murmur.

#val	size[B]	murmur	fibonacci	boncz	crc32	poly 1	poly 4
1	4	10^6	10^6	10^6	10^6	10^6	10^6
2	8	5910.63	30.40	2.00	911.94	14.27	2210.45
4	16	25.02	10.17	1.33	29.54	4.77	28.27
8	32	4.85	4.17	1.14	4.85	1.95	5.16
16	64	2.78	1.92	1.07	3.20	1.33	3.17
32	128	1.88	1.34	1.06	2.10	1.18	1.96
64	256	1.56	1.15	1.03	1.95	1.09	1.60
128	512	1.37	1.07	1.02	1.50	1.05	1.39
256	1024	1.28	1.04	1.01	1.34	1.02	1.31

Table 6.1: Max Q-Error original AKMV (#sketch=100, #insert=10^6).

#ctr	size[B]	murmur	fibonacci	boncz	crc32	poly 1	poly 4
4	4	15.66	1000000.00	18.30	19.06	17.65	74.73
8	8	8.32	500000.00	18.15	9.62	7.96	5.18
16	16	3.56	250000.00	17.88	3.51	4.00	3.12
32	32	2.06	111111.11	17.85	2.67	3.83	2.18
64	64	2.00	52631.58	17.84	1.84	3.68	1.70
128	128	2.00	26315.79	17.81	1.47	3.41	1.50
256	256	1.40	13157.89	17.81	1.43	2.91	1.28
512	512	1.33	6578.95	17.80	1.35	2.75	1.25
1024	1024	1.25	3289.47	17.80	1.24	2.65	1.25

Table 6.2: Max Q-Error HyperLogLog. (#sketch=100, #insert=10^6).

6.3.2 Intersection Size Estimation

In this section, we compare the accuracy of the different intersection size estimation methods based on HLL and AKMV. To this end, we perform the following experiment: Let K and F be sets, initially empty. To fill K and F, we iterate over the values from 1 to 1.1 million. With a 90% chance, a value is

inserted in either (or both) sets K or F. Into which set a value v is inserted depends on two parameters, the probabilities p_isec and p_K. With probability p_isec, v is inserted into both K and F. With probability p_K v is inserted into K. Consequently, with probability 1 - p_isec - p_K v is inserted into F only. Once no more values are inserted into K and F, we build two sketches $X^K, X^F, X \in \{\text{HLL}, \text{AKMV}\}$. Then, an estimate for $|K \cap F|$ is derived from X^K and X^F.

Tables 6.3 to 6.11 show the accuracy of the different intersection size estimation techniques based on both sketches AKMV and HLL. The results of each table are comparable since the size parameters to AKMV and HLL are chosen such that they have the same memory consumption. In the headline of the tables, p_isec and p_K denote the two aforementioned probabilities, given as a per mille number. avg-qe-est and max-qe-est denote the average and maximum q-error of estimator est $\in$ {akmv, bml, ibe}, where *bml* is a shorthand for binomial mean lookup and *ibe* is a shorthand for the technique by Ertl. Note that we do not show results for the inclusion-exclusion principle. However, the inclusion-exclusion principle estimate serves as the start point for ibe. As was shown in [29], ibe estimates are more accurate than inclusion-exclusion principle estimates.

In the following, we discuss some interesting findings from the tables. The q-errors of all estimation techniques reduce in the size of the sketches. For each estimator, we can identify at least one case where this estimator is the most accurate. In most cases though, binomial mean lookup is the most accurate estimator. The estimates of bml and ibe tend to be similar. This comes by no surprise, since both estimates are derived from the same synopsis. While the AKMV estimate tends to be the most inaccurate for a given memory budget, AKMV with twice the memory budget generally holds up with the HLL-based estimators. Especially for larger values of p_isec, this phenomenon can be observed by comparing one of the HLL-based estimates with the AKMV from the table with the next larger memory budget. For instance, for p_isec=200 and p_K=100, we have avg-qe-bml=1.11 and avg-qe-akmv=179.64 in Table 6.8, but avg-qe-akmv=1.26 in Table 6.9.

Based on the results of tables 6.3 to 6.11, we conclude that intersection size estimation via binomial mean lookup using HLLs is the most attractive approach.

p_isec	p_K	avg-qe-akmv	max-qe-akmv	avg-qe-bml	max-qe-bml	avg-qe-ibe	max-qe-ibe
1	50	989.74	1109.00	173.79	1096.00	190.15	1096.00
1	100	989.74	1109.00	306.87	1096.00	334.18	1096.00
1	200	989.74	1109.00	439.67	1096.00	511.66	1096.00
1	500	989.74	1109.00	545.16	1109.00	606.97	1109.00
5	50	4950.30	5243.00	698.40	5170.00	695.79	5170.00
5	100	4950.30	5243.00	1221.52	5170.00	1241.71	5170.00
5	200	4950.30	5243.00	1687.74	5194.00	1943.09	5170.00
5	500	4950.30	5243.00	2238.17	5243.00	2485.66	5243.00
10	50	9900.68	10287.00	1398.67	10247.00	1420.33	10247.00
10	100	9900.68	10287.00	2462.18	10247.00	2522.84	10247.00
10	200	9900.68	10287.00	3318.18	10247.00	3875.29	10247.00
10	500	9900.68	10287.00	4376.18	10287.00	4844.16	10287.00
50	50	49503.37	50400.00	6055.78	50163.00	6017.10	50163.00
50	100	49503.37	50400.00	10343.89	50297.00	10626.31	50297.00
50	200	49503.37	50400.00	13843.57	50233.00	16017.55	50297.00
50	500	49503.37	50400.00	20099.93	50297.00	20530.89	50400.00
100	50	99002.80	100192.00	9644.32	99846.00	9802.49	99838.00
100	100	99002.80	100192.00	15702.74	100159.00	16554.74	99838.00
100	200	99002.80	100192.00	21405.32	99928.00	25258.26	99966.00
100	500	99002.80	100192.00	34395.55	100192.00	31395.05	100192.00
200	50	198002.02	199437.00	10179.02	199339.00	10872.91	199339.00
200	100	198002.02	199437.00	16753.54	199339.00	18873.03	199339.00
200	200	198002.02	199437.00	23504.59	199339.00	28614.30	199174.00
200	500	198002.02	199437.00	48669.90	199372.00	37779.40	199372.00
500	50	495003.92	497088.00	1387.33	495821.00	3614.98	496138.00
500	100	495003.92	497088.00	3169.56	495890.00	5842.71	496484.00
500	200	495003.92	497088.00	7871.79	496720.00	8615.53	496484.00
500	500	495003.92	497088.00	30443.64	496754.00	10991.69	496484.00

Table 6.3: Observed q-errors for AKMVs with 1 values and HLLs with 4 counters.

p_isec	p_K	avg-qe-akmv	max-qe-akmv	avg-qe-bml	max-qe-bml	avg-qe-ibe	max-qe-ibe
1	50	987.50	1109.00	315.87	1096.00	326.56	1096.00
1	100	987.50	1109.00	510.86	1096.00	555.65	1096.00
1	200	987.50	1109.00	561.21	1096.00	797.75	1109.00
1	500	987.50	1109.00	582.70	1109.00	951.15	1109.00
5	50	4897.97	5243.00	1467.41	5170.00	1477.30	5170.00
5	100	4897.97	5243.00	2308.37	5194.00	2505.52	5171.00
5	200	4897.97	5243.00	2520.44	5194.00	3762.76	5194.00
5	500	4897.97	5243.00	2509.30	5243.00	4680.25	5243.00
10	50	9705.25	10287.00	2906.71	10247.00	2927.48	10247.00
10	100	9705.25	10287.00	4585.90	10287.00	5042.25	10287.00
10	200	9705.25	10287.00	4925.02	10247.00	7490.03	10287.00
10	500	9705.25	10287.00	4919.51	10287.00	9198.63	10287.00
50	50	44538.77	50400.00	12545.36	50243.00	13490.90	50243.00
50	100	44538.77	50400.00	17564.06	50297.00	22972.28	50297.00
50	200	44538.77	50400.00	19542.84	50233.00	33846.73	50297.00
50	500	44538.77	50400.00	22036.67	50297.00	41706.78	50400.00
100	50	79795.08	100070.00	17378.33	99878.00	23760.97	99966.00
100	100	79795.08	100070.00	23107.40	100159.00	39355.39	100159.00
100	200	79795.08	100070.00	28145.28	100070.00	57554.31	100159.00
100	500	79795.08	100070.00	34702.53	100192.00	71166.86	100192.00
200	50	125891.37	199437.00	10064.91	199339.00	29659.64	199339.00
200	100	125891.37	199437.00	16258.09	199339.00	49797.85	199339.00
200	200	125891.37	199437.00	28535.66	199197.00	72607.34	199257.00
200	500	125891.37	199437.00	44175.40	199192.00	95875.08	199313.00
500	50	123952.26	496720.00	546.05	496832.00	5099.54	496832.00
500	100	123952.26	496720.00	1041.24	496832.00	10050.18	496832.00
500	200	123952.26	496720.00	2971.25	496832.00	18118.79	496832.00
500	500	123952.26	496720.00	10858.36	496832.00	26980.89	496832.00

Table 6.4: Observed q-errors for AKMVs with 2 values and HLLs with 8 counters.

p_isec	p_K	avg-qe-akmv	max-qe-akmv	avg-qe-bml	max-qe-bml	avg-qe-ibe	max-qe-ibe
1	50	985.85	1109.00	405.98	1096.00	465.85	1096.00
1	100	985.85	1109.00	360.10	1095.00	640.59	1096.00
1	200	985.85	1109.00	415.59	1096.00	726.56	1109.00
1	500	985.85	1109.00	440.63	1109.00	764.53	1109.00
5	50	4850.70	5243.00	1811.01	5171.00	2178.26	5171.00
5	100	4850.70	5243.00	1582.73	5194.00	2968.36	5194.00
5	200	4850.70	5243.00	1785.83	5194.00	3430.45	5194.00
5	500	4850.70	5243.00	1716.64	5243.00	3700.54	5194.00
10	50	9497.68	10287.00	3243.24	10247.00	4219.80	10247.00
10	100	9497.68	10287.00	2980.28	10247.00	5815.96	10287.00
10	200	9497.68	10287.00	3351.75	10247.00	6646.28	10287.00
10	500	9497.68	10287.00	3209.78	10247.00	7104.91	10235.00
50	50	40042.40	50400.00	5720.51	50129.00	14481.88	50233.00
50	100	40042.40	50400.00	9225.72	50163.00	20567.79	50233.00
50	200	40042.40	50400.00	11369.17	50233.00	24706.39	50400.00
50	500	40042.40	50400.00	12810.81	50278.00	27229.20	50400.00
100	50	64568.56	99966.00	4645.37	99825.00	15851.88	99878.00
100	100	64568.56	99966.00	7916.27	99878.00	23180.05	99878.00
100	200	64568.56	99966.00	12597.57	99877.00	30615.88	99878.00
100	500	64568.56	99966.00	17055.81	99966.00	36011.26	100192.00
200	50	80112.00	199372.00	892.84	199079.00	6852.10	199203.00
200	100	80112.00	199372.00	2308.49	199076.00	12040.84	199257.00
200	200	80112.00	199372.00	6200.13	199057.00	17828.00	199203.00
200	500	80112.00	199372.00	13566.37	199054.00	24458.12	199203.00
500	50	32128.61	496720.00	1.24	5.80	50.90	496832.00
500	100	32128.61	496720.00	1.26	10.93	50.92	496832.00
500	200	32128.61	496720.00	1.30	19.36	1.28	12.75
500	500	32128.61	496720.00	446.93	495933.00	397.41	496832.00

Table 6.5: Observed q-errors for AKMVs with 4 values and HLLs with 16 counters.

p_isec	p_K	avg-qe-akmv	max-qe-akmv	avg-qe-bml	max-qe-bml	avg-qe-ibe	max-qe-ibe
1	50	983.07	1109.00	318.09	1094.00	639.32	1109.00
1	100	983.07	1109.00	348.53	1095.00	758.56	1109.00
1	200	983.07	1109.00	394.19	1109.00	797.61	1109.00
1	500	983.07	1109.00	474.55	1109.00	827.47	1109.00
5	50	4788.45	5243.00	1348.82	5171.00	2987.91	5176.00
5	100	4788.45	5243.00	1539.33	5172.00	3545.99	5243.00
5	200	4788.45	5243.00	1755.59	5194.00	3787.26	5194.00
5	500	4788.45	5243.00	1957.18	5243.00	4017.38	5243.00
10	50	9220.57	10287.00	2220.72	10247.00	5654.10	10287.00
10	100	9220.57	10287.00	2724.68	10217.00	6688.29	10287.00
10	200	9220.57	10287.00	3221.60	10247.00	7264.06	10287.00
10	500	9220.57	10287.00	3665.67	10247.00	7717.05	10247.00
50	50	34364.76	50400.00	3190.04	50129.00	13879.24	50400.00
50	100	34364.76	50400.00	5542.39	50173.00	18506.00	50400.00
50	200	34364.76	50400.00	8577.16	50233.00	22607.34	50400.00
50	500	34364.76	50400.00	13137.60	50400.00	26432.79	50400.00
100	50	47342.95	99966.00	843.22	99901.00	8022.39	100192.00
100	100	47342.95	99966.00	2616.74	99732.00	13528.42	100192.00
100	200	47342.95	99966.00	6924.36	99779.00	20755.88	100192.00
100	500	47342.95	99966.00	14773.52	100192.00	26936.56	100192.00
200	50	41046.47	199372.00	21.04	197816.00	516.35	198533.00
200	100	41046.47	199372.00	120.06	198514.00	1805.14	199083.00
200	200	41046.47	199372.00	1130.06	199076.00	4238.91	198995.00
200	500	41046.47	199372.00	7427.55	199079.00	7863.86	199192.00
500	50	4159.49	496720.00	1.15	2.48	1.15	2.58
500	100	4159.49	496720.00	1.16	2.33	1.16	2.64
500	200	4159.49	496720.00	1.18	4.01	1.18	3.48
500	500	4159.49	496720.00	1.24	14.52	1.20	4.24

Table 6.6: Observed q-errors for AKMVs with 8 values and HLLs with 32 counters.

p_isec	p_K	avg-qe-akmv	max-qe-akmv	avg-qe-bml	max-qe-bml	avg-qe-ibe	max-qe-ibe
1	50	976.22	1109.00	144.34	1087.00	666.58	1109.00
1	100	976.22	1109.00	193.63	1095.00	675.47	1096.00
1	200	976.22	1109.00	237.17	1109.00	683.12	1109.00
1	500	976.22	1109.00	360.07	1109.00	697.60	1109.00
5	50	4599.38	5243.00	513.86	5171.00	2969.07	5176.00
5	100	4599.38	5243.00	735.45	5171.00	3083.74	5194.00
5	200	4599.38	5243.00	915.65	5176.00	3147.33	5194.00
5	500	4599.38	5243.00	1362.29	5176.00	3267.26	5243.00
10	50	8507.76	10287.00	751.56	10287.00	5239.31	10287.00
10	100	8507.76	10287.00	1260.92	10287.00	5550.65	10287.00
10	200	8507.76	10287.00	1636.84	10221.00	5813.45	10287.00
10	500	8507.76	10287.00	2419.29	10247.00	5962.09	10247.00
50	50	22767.09	50400.00	368.47	49977.00	6769.56	50243.00
50	100	22767.09	50400.00	1033.47	50002.00	9863.83	50243.00
50	200	22767.09	50400.00	2264.68	50100.00	12784.59	50400.00
50	500	22767.09	50400.00	6378.62	50171.00	14803.21	50400.00
100	50	20363.10	99878.00	11.17	98696.00	1387.53	99584.00
100	100	20363.10	99878.00	130.28	99570.00	3220.43	100192.00
100	200	20363.10	99878.00	694.77	99719.00	6317.65	99928.00
100	500	20363.10	99878.00	4567.67	99872.00	8814.70	99872.00
200	50	6890.96	199372.00	1.14	3.03	1.17	3.98
200	100	6890.96	199372.00	1.20	3.80	1.22	6.47
200	200	6890.96	199372.00	1.27	16.60	140.57	198465.00
200	500	6890.96	199372.00	615.12	198563.00	397.58	198629.00
500	50	1.30	8.41	1.12	1.81	1.12	1.76
500	100	1.30	8.41	1.12	1.73	1.13	1.84
500	200	1.30	8.41	1.12	1.87	1.14	1.88
500	500	1.30	8.41	1.15	2.30	1.15	2.15

Table 6.7: Observed q-errors for AKMVs with 16 values and HLLs with 64 counters.

p_isec	p_K	avg-qe-akmv	max-qe-akmv	avg-qe-bml	max-qe-bml	avg-qe-ibe	max-qe-ibe
1	50	959.07	1109.00	43.60	1056.00	658.11	1109.00
1	100	959.07	1109.00	70.08	1087.00	672.86	1109.00
1	200	959.07	1109.00	90.78	1082.00	686.61	1109.00
1	500	959.07	1109.00	190.63	1109.00	699.25	1109.00
5	50	4256.18	5243.00	44.77	5092.00	2784.83	5184.00
5	100	4256.18	5243.00	119.19	5131.00	2892.14	5184.00
5	200	4256.18	5243.00	195.11	5176.00	3036.50	5184.00
5	500	4256.18	5243.00	504.70	5194.00	3144.21	5243.00
10	50	7163.89	10287.00	43.76	10060.00	4565.64	10287.00
10	100	7163.89	10287.00	145.81	10121.00	4965.43	10287.00
10	200	7163.89	10287.00	292.07	10181.00	5308.71	10287.00
10	500	7163.89	10287.00	792.18	10235.00	5446.12	10247.00
50	50	10146.75	50400.00	1.60	10.11	2362.94	50129.00
50	100	10146.75	50400.00	1.84	108.08	3972.82	50129.00
50	200	10146.75	50400.00	81.15	49856.00	6213.29	50163.00
50	500	10146.75	50400.00	1037.27	50139.00	7038.14	50171.00
100	50	4000.89	99781.00	1.26	3.10	90.37	99386.00
100	100	4000.89	99781.00	1.38	4.90	416.82	99358.00
100	200	4000.89	99781.00	11.37	99177.00	853.16	99570.00
100	500	4000.89	99781.00	219.39	99185.00	1426.90	99707.00
200	50	179.64	198509.00	1.11	1.58	1.11	2.04
200	100	179.64	198509.00	1.16	2.21	1.14	2.50
200	200	179.64	198509.00	1.19	3.05	1.16	2.84
200	500	179.64	198509.00	1.27	6.19	1.18	3.04
500	50	1.18	3.39	1.07	1.36	1.10	1.45
500	100	1.18	3.39	1.07	1.41	1.10	1.54
500	200	1.18	3.39	1.08	1.49	1.11	1.57
500	500	1.18	3.39	1.09	1.83	1.11	1.60

Table 6.8: Observed q-errors for AKMVs with 32 values and HLLs with 128 counters.

p_isec	p_K	avg-qe-akmv	max-qe-akmv	avg-qe-bml	max-qe-bml	avg-qe-ibe	max-qe-ibe
1	50	928.59	1109.00	46.89	63.48	567.16	1109.00
1	100	928.59	1109.00	66.35	117.67	537.28	1109.00
1	200	928.59	1109.00	71.48	1021.00	490.81	1109.00
1	500	928.59	1109.00	119.87	1077.00	428.80	1096.00
5	50	3643.08	5243.00	10.12	13.29	2096.94	5184.00
5	100	3643.08	5243.00	14.02	23.56	2113.42	5184.00
5	200	3643.08	5243.00	17.42	5011.00	1939.14	5176.00
5	500	3643.08	5243.00	89.19	5137.00	1696.53	5184.00
10	50	5170.78	10287.00	5.54	7.39	2863.48	10287.00
10	100	5170.78	10287.00	7.46	11.71	3038.18	10287.00
10	200	5170.78	10287.00	8.95	9931.00	2880.56	10287.00
10	500	5170.78	10287.00	99.15	10184.00	2491.90	10287.00
50	50	1956.45	50400.00	1.85	2.46	159.85	49913.00
50	100	1956.45	50400.00	2.24	3.45	496.59	50107.00
50	200	1956.45	50400.00	2.33	4.13	710.02	50054.00
50	500	1956.45	50400.00	17.61	49371.00	672.33	49983.00
100	50	169.69	99511.00	1.38	1.83	1.13	2.47
100	100	169.69	99511.00	1.58	2.08	1.16	2.53
100	200	169.69	99511.00	1.62	2.57	20.99	99201.00
100	500	169.69	99511.00	11.70	99242.00	11.09	98903.00
200	50	1.26	6.44	1.14	1.43	1.08	1.50
200	100	1.26	6.44	1.24	1.60	1.10	1.69
200	200	1.26	6.44	1.27	1.70	1.11	1.98
200	500	1.26	6.44	1.27	1.95	1.12	1.99
500	50	1.12	1.91	1.05	1.27	1.09	1.38
500	100	1.12	1.91	1.04	1.24	1.09	1.38
500	200	1.12	1.91	1.06	1.29	1.09	1.43
500	500	1.12	1.91	1.06	1.37	1.09	1.44

Table 6.9: Observed q-errors for AKMVs with 64 values and HLLs with 256 counters.

p_isec	p_K	avg-qe-akmv	max-qe-akmv	avg-qe-bml	max-qe-bml	avg-qe-ibe	max-qe-ibe
1	50	866.43	1109.00	50.38	63.22	629.61	1109.00
1	100	866.43	1109.00	98.87	124.28	634.39	1109.00
1	200	866.43	1109.00	123.08	187.73	654.23	1096.00
1	500	866.43	1109.00	137.76	267.20	686.33	1109.00
5	50	2600.80	5243.00	10.81	12.86	2145.91	5171.00
5	100	2600.80	5243.00	20.47	24.22	2378.56	5184.00
5	200	2600.80	5243.00	25.41	37.91	2592.01	5184.00
5	500	2600.80	5243.00	27.97	52.25	2812.80	5194.00
10	50	2700.04	10287.00	5.88	7.05	2607.00	10287.00
10	100	2700.04	10287.00	10.67	12.84	3330.69	10287.00
10	200	2700.04	10287.00	13.17	20.13	3842.61	10222.00
10	500	2700.04	10287.00	14.44	26.01	4366.48	10247.00
50	50	80.72	49923.00	1.92	2.30	15.98	49524.00
50	100	80.72	49923.00	2.84	3.39	199.38	50133.00
50	200	80.72	49923.00	3.42	4.61	556.52	50041.00
50	500	80.72	49923.00	3.52	5.42	1021.51	50008.00
100	50	1.27	6.64	1.43	1.66	1.09	1.72
100	100	1.27	6.64	1.86	2.25	1.11	1.73
100	200	1.27	6.64	2.20	2.84	1.13	2.20
100	500	1.27	6.64	2.17	2.98	11.74	98919.00
200	50	1.16	2.30	1.18	1.39	1.06	1.36
200	100	1.16	2.30	1.37	1.59	1.06	1.45
200	200	1.16	2.30	1.58	1.95	1.08	1.52
200	500	1.16	2.30	1.52	2.00	1.08	1.54
500	50	1.08	1.57	1.04	1.18	1.04	1.22
500	100	1.08	1.57	1.07	1.24	1.04	1.28
500	200	1.08	1.57	1.17	1.33	1.04	1.25
500	500	1.08	1.57	1.21	1.44	1.05	1.30

Table 6.10: Observed q-errors for AKMVs with 128 values and HLLs with 512 counters.

p_isec	p_K	avg-qe-akmv	max-qe-akmv	avg-qe-bml	max-qe-bml	avg-qe-ibe	max-qe-ibe
1	50	766.52	1109.00	50.62	59.62	530.39	1109.00
1	100	766.52	1109.00	100.09	121.08	504.76	1109.00
1	200	766.52	1109.00	195.38	235.65	453.53	1109.00
1	500	766.52	1109.00	197.53	278.70	403.03	1096.00
5	50	1396.48	5243.00	10.87	12.40	1258.35	5184.00
5	100	1396.48	5243.00	20.72	23.60	1386.78	5184.00
5	200	1396.48	5243.00	39.69	44.96	1356.13	5171.00
5	500	1396.48	5243.00	40.09	51.68	1194.06	5194.00
10	50	771.29	10182.00	5.91	6.75	882.38	10222.00
10	100	771.29	10182.00	10.79	12.18	1249.23	10222.00
10	200	771.29	10182.00	20.25	22.73	1297.54	10213.00
10	500	771.29	10182.00	20.52	25.72	1112.88	10182.00
50	50	1.28	6.25	1.93	2.20	1.11	1.81
50	100	1.28	6.25	2.86	3.21	6.10	49607.00
50	200	1.28	6.25	4.69	5.23	6.12	49631.00
50	500	1.28	6.25	4.76	5.91	6.11	49319.00
100	50	1.18	2.94	1.43	1.61	1.06	1.41
100	100	1.18	2.94	1.87	2.11	1.08	1.53
100	200	1.18	2.94	2.74	3.04	1.09	1.61
100	500	1.18	2.94	2.84	3.51	1.10	1.68
200	50	1.11	2.01	1.18	1.32	1.04	1.21
200	100	1.11	2.01	1.37	1.54	1.04	1.28
200	200	1.11	2.01	1.75	1.99	1.05	1.32
200	500	1.11	2.01	1.92	2.30	1.06	1.42
500	50	1.06	1.33	1.02	1.14	1.04	1.16
500	100	1.06	1.33	1.06	1.18	1.04	1.18
500	200	1.06	1.33	1.16	1.28	1.04	1.19
500	500	1.06	1.33	1.36	1.50	1.04	1.18

Table 6.11: Observed q-errors for AKMVs with 256 values and HLLs with 1024 counters.

Chapter 7

Two-way Joins

7.1 Introduction

This chapter presents a new join size estimation technique. Note that join size estimation is critical in query optimization, since estimation errors propagate exponentially through joins and lead to slow query plans [40]. In the world of relational database systems, key/foreign-key joins are frequent and hence play an important role [10, 67]. The main difficulty in key/foreign-key join size estimation arises from selection predicates that are applied to the relations before they are joined [23, 86].

The problem scope considered in this chapter is to estimate the size of an equi-join of two filtered relations R and S, i.e.,

$$|\sigma_{p_R}(R) \bowtie_{p_J} \sigma_{p_S}(S)|,$$

where p_J is a key/foreign-key equality join predicate. For now, assume p_R and p_S both are range predicates. Note that point predicates (=) and half-open predicates are special range predicates. We will see later that p_R and p_S can be generalized to more complex types of predicates. The above join scenario occurs frequently in relational database systems, e.g., in a data warehouse where a fact table joins with a dimension table after a predicate has been applied to each table.

Many existing join size estimation techniques rely on the well-established formula

$$|\sigma_{p_R}(R) \bowtie_{p_J} \sigma_{p_S}(S)| = s_{p_J} \cdot |\sigma_{p_R}(R) \times \sigma_{p_S}(S)|,$$

where s_{p_J} denotes the selectivity of p_J, and $|\sigma_{p_R}(R) \times \sigma_{p_S}(S)|$ denotes the cross product size of the filtered relations, which equals $|\sigma_{p_R}(R)| \cdot |\sigma_{p_S}(S)|$. To estimate join sizes, several techniques were developed. The simplest [78] relies on (unrealistic) independence and uniformity assumptions. Sampling for join size estimation [23, 86] is versatile but selective selection predicates can cause inaccurate join size estimates [88]. A different approach are join size

sketches [13, 25], but sketches are built *after* selection predicates have been applied.

In this work, we consider a different join size model, which has no notion of join selectivity. In pictures, joins are often illustrated as the intersection of join attributes in a Venn diagram. Yet a join can be much larger than the intersection size of the join attributes. For join attributes $R.K$ and $S.F$, the relationship between join size and intersection size is expressed in a formula by Allen Van Gelder [85], which is the basis of our work,

$$|\sigma_{p_R}(R) \bowtie_{R.K=S.F} \sigma_{p_S}(S)| = \alpha \cdot \mathcal{I},$$

where $\mathcal{I}$ is the intersection size of the filtered join attribute values, i.e. $\mathcal{I} \coloneqq |\pi_K(\sigma_{p_R}(R)) \cap \pi_F(\sigma_{p_S}(S))|$, which is a lower bound for the join size, and α is called the *average multiplicity* [85]. Let $R' = \sigma_{p_R}(R)$ and $S' = \sigma_{p_S}(S)$. Then α is the average number of times a join attribute value occurs in the join result, that is,

$$\begin{aligned}\alpha &\coloneqq \tfrac{1}{\mathcal{I}} \textstyle\sum_{v \in R'.K \cap S'.F} |\sigma_{K=v}(R')| \cdot |\sigma_{F=v}(S')| \\ &= \frac{|R' \bowtie_{R'.K=S'.F} S'|}{|R'.K \cap S'.F|}.\end{aligned}$$

For a non-empty join, the average multiplicity α must be greater than or equal to 1. We define $\alpha = 0$ for joins with empty intersection size to prevent division by 0.

Applying the definitions of join selectivity, average multiplicity, and intersection size to a join $R \bowtie_{R.K=S.F} S$ indeed yields the true join size $|R \bowtie_{R'.K=S'.F} S|$ for both the selectivity and the multiplicity based formula.

To illustrate the formula $\alpha \cdot \mathcal{I}$, consider the following example tables

R

K	B
1	2
2	7
3	3
4	1
5	2

S

F	Z
1	3
2	10
2	2
2	5
2	8
3	7
3	8
4	2
5	5

and the example query

$$\sigma_{B \geq 3}(R) \bowtie_{R.K=S.F} \sigma_{4 \leq Z \leq 10}(S).$$

Let $R' := \sigma_{B \geq 3}(R)$ and $S' := \sigma_{4 \leq Z \leq 10}(S)$. The tuples shaded in gray are the ones that qualify the selection predicate. Then, the intersection size of the join attribute values is

$$\mathcal{I} = |R'.K \cap S'.F| = |\{2, 3\} \cap \{2, 3, 5\}| = 2.$$

R	Table with schema {K `int primary key`, B `int`}.
S	Table with schema {F `int foreign key references` R, Z `int`}.
p_R	A range selection predicate $c_1 \leq R.B \leq c_2$, where c_1 and c_2 are query-specific constants.
p_S	A range selection predicate $d_1 \leq S.Z \leq d_2$, where d_1 and d_2 are query-specific constants.
p_J	The join predicate $R.K = S.F$.
π^D_X	Projection on attribute X with duplicate removal.
HLL	Abbreviation for HyperLogLog sketch.
$\mathcal{B}_T$	A bucket sketch for table T.
Freq	Counts number of inserts into a bucket.
CtrFreq	Counter Frequency. Optional field in buckets of a bucket sketch. Approximates multiplicity of keys.
FK	Foreign key.
Key table	Table with the key attribute of a key/FK join.
FK table	Table with the FK attribute of a key/FK join.
NODV	Number of distinct values.
CAMA	Constant Average Multiplicity Assumption

Table 7.1: Symbols and abbreviations.

For the average multiplicity, we have

$$\alpha = \frac{1}{\mathcal{I}} \sum_{v \in R'.K \cap S'.F} |\sigma_{K=v}(R')| \cdot |\sigma_{F=v}(S')|,$$

which gives $\frac{1}{2}(3+2) = 2.5$. Finally, the join size is

$$\alpha \cdot \mathcal{I} = 2.5 \cdot 2 = 5,$$

which is indeed the true join size.

As stated in the problem scope, we present an estimation method for the key/foreign-key join of two filtered relations. We apply the formula $\alpha \cdot \mathcal{I}$. Therefore, we present several formulas to estimate the average multiplicity α. We evaluate recent developments in the area of intersection size estimation which do not consider predicates [29, 68]. However, a data structure we designed, called the *bucket sketch*, allows one to consider simple predicates and estimate the average multiplicity α and the intersection size $\mathcal{I}$ of a join of two filtered relations.

The remainder of this chapter is structured as follows: Section 7.2 presents our join size estimation method. Then, in Section 7.3, we evaluate this method and compare it to various other estimators. Finally, in Section 7.4, we summarize the contents of this chapter.

Throughout the chapter, we use the relations R and S, the selection predicates p_R and p_S, and the join predicate p_J. All are given in Table 7.1.

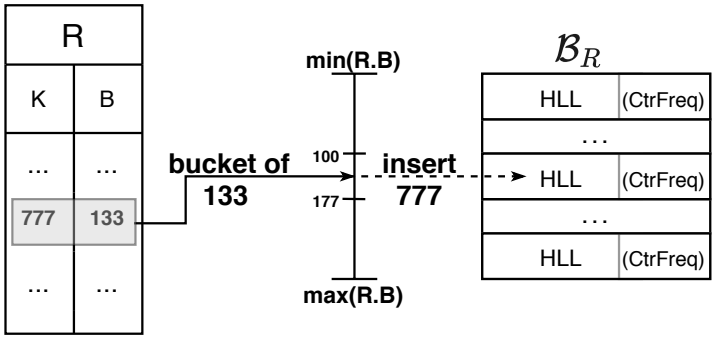


Figure 7.1: Bucket sketch $\mathcal{B}_R$ is constructed for relation R. For join size estimation, a bucket sketch $\mathcal{B}_S$ for relation S must be constructed as well.

7.2 Join Size Estimation with Bucket Sketches

As presented in the introduction, our join size estimation technique allows to estimate the size of the key/foreign-key join $\sigma_{p_R}(R) \bowtie_{p_J} \sigma_{p_S}(S)$ via the formula $\alpha \cdot \mathcal{I}$, where $\mathcal{I}$ is the intersection size of the join attribute values of p_J in $\sigma_{p_R}(R)$ and $\sigma_{p_S}(S)$, and α is the average multiplicity of the intersection values in the join.

The following sections are structured as follows: Section 7.2.1 presents the bucket sketch, a data structure we construct offline for relations R and S, as illustrated in Figure 7.1. For a given join query, the high level procedure to estimate a join size is illustrated in Figure 7.2. Bucket sketches handle the selection predicates p_R and p_S. Via bucket sketches, we also obtain intersection size estimates subject to filter predicates, as presented in Section 7.2.1.4. Section 7.2.2 presents formulas to estimate the average multiplicity α. Section 7.2.3 analyzes the estimation error and presents confidence intervals. Section 7.2.4 discusses generalizations of our approach.

7.2.1 Bucket Sketch

We present the *bucket sketch*, a data structure used for average multiplicity and intersection size estimation. A bucket sketch must be built for relations R and S, respectively. This subsection is structured as follows: Section 7.2.1.1 presents the structure of a bucket sketch, Section 7.2.1.2 shows how to construct a bucket sketch from an input relation. Sections 7.2.1.3 and 7.2.1.4 show how to estimate certain expression sizes from one and two bucket sketches, respectively.

7.2.1.1 Structure

A bucket sketch $\mathcal{B}_T$ is defined in terms of a non-overlapping partitioning of a relation T along some *partitioning attribute* $T.G$. Each partition defines the range of a *bucket* in the bucket sketch $\mathcal{B}_T$. Each bucket contains an HLL sketch

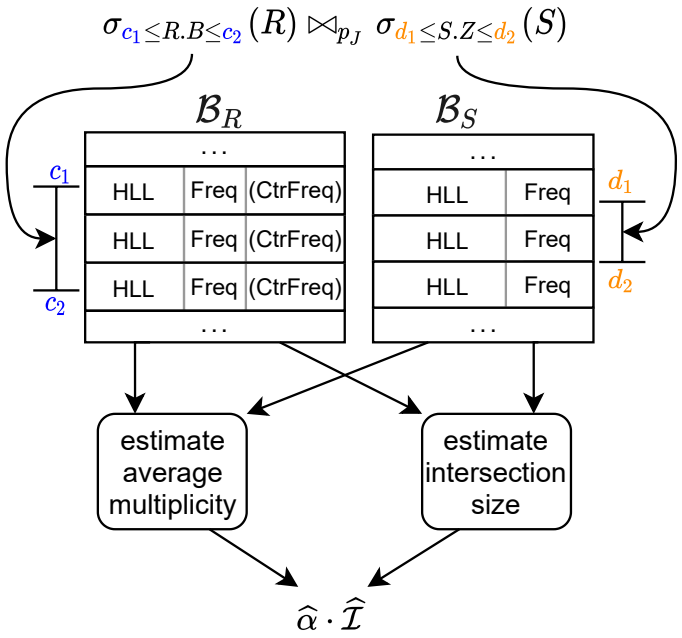


Figure 7.2: Join size estimation using $\mathcal{B}_R$ and $\mathcal{B}_S$ to derive an estimated average multiplicity $\widehat{\alpha}$ and estimated intersection size $\widehat{\mathcal{I}}$ for query $\sigma_{c_1 \leq R.B \leq c_2}(R) \bowtie_{p_J} \sigma_{d_1 \leq S.Z \leq d_2}(S)$.

that is used to approximately count the NODV of an attribute $T.J$, to which we refer as the *HLL attribute*. We will later discuss that any type of distinct count sketch can be used, e.g., AKMV. However, as we have seen in the previous chapter, Chapter 6, and will see again in the evaluation section of this chapter, the best accuracy is achieved with HLL. Instead of a single attribute, $T.J$ can be an attribute set, which corresponds to a conjunctive join predicate. The only requirement for $T.J$ is to be hashable. In addition to the HLL sketch, each bucket contains a *Freq* field that counts the number of tuples that belong to that bucket. Note that G and J can either be the same or differ, but must belong to the same relation T. In Section 7.3.7, we show that, using the appropriate intersection size estimation technique, HLL sketches with as few as 8 counters suffice – the size of a single 64-bit pointer!

As indicated by the parentheses in Figure 7.1, the buckets of a key table bucket sketch $\mathcal{B}_R$ optionally contain the *CtrFreq* field (short for *counter frequency*), which is used to store the sum of the frequencies of the matching FKs. More formally, let $P \subseteq R$ be the partition that defines the range of some bucket $\mathcal{B}_R[i]$ of $\mathcal{B}_R$. Then, $\mathcal{B}_R[i].CtrFreq = |P \bowtie_{p_J} S|$. Note that the frequency of an FK attribute value v is the same as v's multiplicity in an unfiltered join.

For the bucket sketch, we use the equi-width partitioning scheme, defined as $\frac{|dom|}{n}$, where $|dom| = \max(T.G) - \min(T.G) + 1$ and n is the number of buck-

Algorithm 1 (Gray lines (2, 7, 14) are optional.)

```
ConstructSketches(R, n_R, S, n_S)
    Input:  Relations R and S,
            number of buckets n_R, n_S in bucket sketches
    Output: Bucket sketches B_R and B_S
 1  Let B_S be a bucket sketch with n_S buckets
 2  Let CtrFreqHT be a hash table
 3  for each tuple s ∈ S
 4      idx = GetBucketIndex(s.Z)
 5      B_S[idx].HLL.insert(s.F)
 6      B_S[idx].Freq += 1
 7      CtrFreqHT[s.F] += 1 // initially zero
 8
 9  Let B_R be a bucket sketch with n_R buckets
10  for each tuple r ∈ R
11      idx = GetBucketIndex(r.B)
12      B_R[idx].HLL.insert(r.K)
13      B_R[idx].Freq += 1
14      B_R[idx].CtrFreq += CtrFreqHT[r.K]
15  return B_R, B_S
```

ets [48]. If the min and max value of $T.G$ are known, e.g., in the statistics catalog of the database system, we construct a bucket sketch in a single pass over the data as in [14]. Otherwise, we perform an additional pre-processing pass over the data to determine the min and max value, adding up to two passes.

To determine the bucket $\mathcal{B}_T[idx_v]$ corresponding to a value $v \in T.G$, we could store the bucket boundaries and perform a binary search. Instead, we apply a direct access method and avoid storing the bucket boundaries: Say our memory budget allows for $n_{max} = 128$ buckets, where n_{max} is always a power of two. If $|dom| > n_{max}$, we choose the number of buckets $n := n_{max}$ and find the bucket of a value $v \in T.G$ as

$$idx_v = (v - \min(T.G)) \gg shiftby,$$

where $\gg$ denotes logical bit shift right and *shiftby* is the number of bits we need for the domain minus the number of bits we have for indexing, i.e., $shiftby := \lceil \log_2(|dom|) \rceil - \log_2(n)$.

In case $|dom| \leq n_{max}$, we have $n := |dom|$ buckets and simply compute $idx_v = v - \min(T.G)$. This is beneficial for attributes with small domains like `nationkey` in TPC-H.

7.2.1.2 Construction

The bucket sketches $\mathcal{B}_R$ and $\mathcal{B}_S$ of relations R and S are built offline as described in Algorithm 1. The gray lines (2, 7, 14) are optional and must be executed only if *CtrFreq* is used in $\mathcal{B}_R$. Upon creation of a bucket sketch, all fields are initialized to zero. To populate $\mathcal{B}_S$, iterate over S (line 3). For each tuple, find the index corresponding to $s.Z$ and insert the FK value $s.F$ into the

Algorithm 2

GetMergedSketches($\mathcal{B}$, Q)

Input: A bucket sketch $\mathcal{B}$,
a query Q, selects from table $\mathcal{B}$ is built on
Output: Two merged HLL sketches

```
1  Let H_⌊∪⌋, H_⌈∪⌉ be HLLs with m counters set to zero
2  for each bucket ∈ B
3      if IsContained(bucket, Q)
4          for i = 0 to m−1
5              H_⌊∪⌋[i] = max(H_⌊∪⌋[i], bucket.HLL[i])
6      if IsIntersected(bucket, Q)
7          for i = 0 to m−1
8              H_⌈∪⌉[i] = max(H_⌈∪⌉[i], bucket.HLL[i])
9  return H_⌊∪⌋, H_⌈∪⌉
```

HLL. Additionally, the insert frequency is incremented. Line 7 (gray) uses the temporary hash table created in line 2 to count the frequency of each FK.

The bucket sketch $\mathcal{B}_R$ for relation R is constructed in the same fashion as $\mathcal{B}_S$. For each tuple in R (line 10), we find the corresponding bucket, insert the key value $r.K$ into the HLL and increment the insert frequency. In line 14, $\mathcal{B}_R[idx].CtrFreq$ sums the frequencies of the FKs corresponding to $r.K$ values in this bucket. Note that double counting entries of *CtrFreqHT* is impossible since the key attribute values in K are unique.

7.2.1.3 One Bucket Sketch: Estimating Distinct Counts after Selections

It is not difficult to see that a bucket sketch $\mathcal{B}_T$, partitioned by attribute $T.G$ and with $T.J$ as the HLL attribute, can be used to estimate

$$\mathcal{D} \coloneqq |\pi_J^D((\sigma_{p_T}(T)))|,$$

where π^D denotes projection with duplicate removal on $T.J$ and p_T is a selection predicate on $T.G$. That is, one can estimate the NODV in $T.J$ after a selection on $T.G$. To process a selection predicate over multiple buckets in $\mathcal{B}_T$, the HLLs in the buckets must be *mergeable* [11]. Mergeability is given if two HLLs H_1 and H_2 for multisets M_1 and M_2, respectively, can be used to derive an HLL $H_\cup$ such that $H_\cup = H_{M_1 \cup M_2}$, where equality of HLLs means that all their counters have the same value. As shown in [29], if H_1, H_2, and $H_\cup$ have m counters each, $H_\cup[i]$ is found by setting $H_\cup[i] = \max(H_1[i], H_2[i])$, where $0 \leq i \leq m-1$ and $H[i]$ references counter i in H.

We repeatedly apply this merge operation in Algorithm 2. The inputs are a bucket sketch $\mathcal{B}$ and a query Q. Q holds the selection predicate p_T from above. The outputs are the two HLLs $H_{\lfloor\cup\rfloor}$ and $H_{\lceil\cup\rceil}$ created in line 1. Starting in line 2, we loop over the buckets in $\mathcal{B}$ and merge $H_{\lfloor\cup\rfloor}$ with the HLL of all buckets contained by predicate p_T. Similarly, we merge $H_{\lceil\cup\rceil}$ with all buckets intersected by p_T. Finally, $H_{\lfloor\cup\rfloor}$ and $H_{\lceil\cup\rceil}$ are returned. A note on implementation details:

Algorithm 3

EstimateJoinSize($\mathcal{B}_R, \mathcal{B}_S, Q, R, S$)

Input: Bucket sketches $\mathcal{B}_R$, $\mathcal{B}_S$,
a join query Q,
key table R and FK table S
Output: Join size estimate

```
1  H^R_⌊∪⌋, H^R_⌈∪⌉ = GetMergedSketches(B_R, Q)
2  H^S_⌊∪⌋, H^S_⌈∪⌉ = GetMergedSketches(B_S, Q)
3
4  // Estimate average multiplicity α̂, details in Sec. 7.2.2
5  if PredOnKeyTableOnly(Q)
6      if not CtrFreqUsed(B_R)
7          α̂ = Formula_7.3(S)
8      else
9          α̂ = Formula_7.4(B_R)
10 else if PredOnBothTables(Q) ∨
           PredOnFKTableOnly(Q)
11     α̂ = Formula_7.5(B_S)
12 else // No selection predicates
13     return |S|
14
15 // Estimate intersection size Î
16 Î = sqrt(|H^R_⌊∪⌋ ∩ H^S_⌊∪⌋| · |H^R_⌈∪⌉ ∩ H^S_⌈∪⌉|)
17
18 return α̂ · Î // Return join size estimate
```

(1) Instead of looping over all buckets, the relevant buckets can be extracted from the query range. (2) Intel's SIMD instruction `_mm_max_pu8` merges 8 HLL counters at once.

Ignoring HLL estimation errors, $|H_{\lfloor\cup\rfloor}|$, i.e., the NODV as estimated by $H_{\lfloor\cup\rfloor}$, is a lower bound for $\mathcal{D}$. Equivalently, $|H_{\lceil\cup\rceil}|$ is an upper bound for $\mathcal{D}$. Our estimate is the geometric mean

$$\widehat{\mathcal{D}} \coloneqq \sqrt{|H_{\lfloor\cup\rfloor}| \cdot |H_{\lceil\cup\rceil}|}, \tag{7.1}$$

since the geometric mean of $|H_{\lfloor\cup\rfloor}|$ and $|H_{\lceil\cup\rceil}|$ minimizes the factor by which we under- or overestimate at most.

7.2.1.4 Two Bucket Sketches: Estimating Join Sizes after Selections

In this section, we outline how to estimate the join size

$$\mathcal{J} \coloneqq |\sigma_{p_R}(R) \bowtie_{p_J} \sigma_{p_S}(S)|,$$

where the relations and predicates are defined as in Table 7.1. Figure 7.2 gives an overview. First, two bucket sketches $\mathcal{B}_R$ and $\mathcal{B}_S$ are used to process the range predicates, then average multiplicity and intersection size for the filtered relations are estimated as $\widehat{\alpha}$ and $\widehat{\mathcal{I}}$, respectively. Finally, the join size estimate is $\widehat{J} \coloneqq \widehat{\alpha} \cdot \widehat{\mathcal{I}}$.

The high-level procedure to estimate a join size is shown in Algorithm 3. Here, we also consider the case where not both relations are filtered before joining. The inputs to Algorithm 3 are bucket sketches $\mathcal{B}_R$, $\mathcal{B}_S$, and a join query Q, which also specifies the range predicates. The relations R and S are handed in for ease of notation. First, the merged HLL sketches $H^R_{\lfloor\cup\rfloor}, H^R_{\lceil\cup\rceil}$ and $H^S_{\lfloor\cup\rfloor}, H^S_{\lceil\cup\rceil}$ are computed by calling Algorithm 2. Then, the average multiplicity is estimated, which Section 7.2.2 discusses in detail. The estimate depends on which of the relations R and S are filtered. If only the key table R is filtered (line 5) and $\mathcal{B}_R$ has no *CtrFreq* counters, then $\widehat{\alpha}$ is the result of Formula (7.3), which only relies on rudimentary table and column statistics of the FK table S. If the optional *CtrFreq* variable in the buckets of $\mathcal{B}_R$ is used, then Formula (7.4) determines $\widehat{\alpha}$. In the two cases where either both tables or only the FK table is filtered (line 10), $\widehat{\alpha}$ is determined by Formula (7.5). If no relation is filtered, the join size (estimate) is $|S|$, since in a key/FK join without selection predicates each FK finds one matching key and, hence, the join size is always the size of the FK table.

Next, an estimate for the intersection size

$$\mathcal{I} = |\pi_K(\sigma_{p_R}(R)) \cap \pi_F(\sigma_{p_S}(S))|$$

is derived. Let H_{M_1}, H_{M_2} be HLLs for multisets M_1, M_2. We denote by $|H_{M_1} \cap H_{M_2}|$ an estimate for $|M_1 \cap M_2|$. To compute $|H_{M_1} \cap H_{M_2}|$, recall the methods from Chapter 6, in particular Section 6.2.2. Irrespective of the method, in line 16 in Algorithm 3, we derive the intersection size estimate $\widehat{\mathcal{I}}$ from the four merged HLL sketches $H^R_{\lfloor\cup\rfloor}$, $H^R_{\lceil\cup\rceil}$, $H^S_{\lfloor\cup\rfloor}$, and $H^S_{\lceil\cup\rceil}$ computed in lines 1 and 2. Ignoring the estimation errors, $|H^R_{\lfloor\cup\rfloor} \cap H^S_{\lfloor\cup\rfloor}|$ is a lower bound for $\mathcal{I}$ since only the two merged HLLs from buckets *contained* by p_R and p_S are intersected. $|H^R_{\lceil\cup\rceil} \cap H^S_{\lceil\cup\rceil}|$ is an upper bound for $\mathcal{I}$ since the two merged HLLs from all buckets *intersected* by p_R and p_S are intersected. Their geometric mean minimizes the multiplicative factor by which we under- or overestimate at most and hence

$$\widehat{\mathcal{I}} \coloneqq \sqrt{|H^R_{\lfloor\cup\rfloor} \cap H^S_{\lfloor\cup\rfloor}| \cdot |H^R_{\lceil\cup\rceil} \cap H^S_{\lceil\cup\rceil}|}. \tag{7.2}$$

Finally, we have $\widehat{\mathcal{J}} = \widehat{\alpha} \cdot \widehat{\mathcal{I}}$ for the estimated join size, which is returned in line 18.

7.2.2 Average Multiplicity Estimation

In this section, we present formulas to derive $\widehat{\alpha}$, an estimate for the average multiplicity α. We consider the queries $|\sigma_{p_R}(R) \bowtie S|$, $|R \bowtie \sigma_{p_S}(S)|$, and $|\sigma_{p_R}(R) \bowtie \sigma_{p_S}(S)|$.

7.2.2.1 Case $|\sigma_{p_R}(R) \bowtie_{R.K=S.F} S|$

In relational algebra, we have that $|R \bowtie_{R.K=S.F} S| = |S|$, since each tuple in the FK table S finds one match in the key table R. It follows that the average

multiplicity is $\frac{|S|}{|\pi^D_{S.F}(S)|}$. For $|\sigma_{p_R}(R) \bowtie_{R.K=S.F} S|$, our first estimate builds on the assumption that, on average, p_R does not change α. We call this the *constant average multiplicity assumption* (CAMA) and it induces the estimate

$$\widehat{\alpha}_{Base} := \frac{|S|}{|\pi^D_{S.F}(S)|}, \tag{7.3}$$

where $S.F$ denotes the FK join attribute. The accuracy of the estimate $\widehat{\alpha}_{Base}$ depends on the relationship between the average multiplicity α and the predicate p_R. This relationship is influenced by the correlation between the join attribute $R.K$ and the attribute p_R is applied to, and the degree of skew in the FK attribute $S.F$.

We can discard CAMA if the bucket sketch $\mathcal{B}_R$ provides the optional *CtrFreq* field. Let P_R be the set of indices of buckets in $\mathcal{B}_R$ intersected by the selection predicate p_R. Then, we obtain an estimate as

$$\widehat{\alpha}_{CtrFreq} := \frac{\sum_{i \in P_R} \mathcal{B}_R[i].CtrFreq}{\sum_{i \in P_R} \mathcal{B}_R[i].Freq}. \tag{7.4}$$

The denominator approximates $|\pi^D_{S.F}(\sigma(R)p_R \bowtie S)|$ well since each key $R.K$ is inserted into $\mathcal{B}_R$ only once. Only in cases where many keys in $R.K$ have no matching FK, it is likely to obtain an underestimate.

7.2.2.2 Case $|R \bowtie_{p_J} \sigma_{p_S}(S)|$

We derive the average multiplicity based on properties of referential integrity. Recall that, for key/foreign-key joins, we have that $|R \bowtie_{p_J} S| = |S|$. Since p_S does not harm referential integrity, all values in $\sigma_{p_S}(S)$ still find one match in R. It follows $|R \bowtie_{p_J} \sigma_{p_S}(S)| = |\sigma_{p_S}(S)|$.

Clearly, we could stop here and estimate the selectivity of p_S. However, we show that an accurate estimate for α can be derived, and this estimate will be used in Section 7.2.2.3.

From the above discussion it follows that the average multiplicity is $\alpha = \frac{|R \bowtie_{p_J} \sigma_{p_S}(S)|}{|\pi_K(R) \cap \pi_F(\sigma_{p_S}(S))|} = \frac{|\sigma_{p_S}(S)|}{|\pi^D_F(\sigma_{p_S}(S))|}$. Let P_S be the set of indices of buckets in $\mathcal{B}_S$ intersected by the selection predicate p_S. The estimate $\widehat{\alpha}_{FK}$ is simply

$$\widehat{\alpha}_{FK} := \frac{\widehat{|\sigma_{p_S}(S)|}}{\widehat{|\pi^D_F(\sigma_{p_S}(S))|}} := \frac{\sum_{i \in P_S} \mathcal{B}_S[i].Freq}{|H^S_{\lceil \cup \rceil}|}. \tag{7.5}$$

The denominator is derived by merging HLLs in $\mathcal{B}_S$. For the numerator, instead of using *Freq*, any single relation cardinality estimator is applicable. Perfect estimates for $|\sigma_{p_S}(S)|$ and $|\pi^D_F(\sigma_{p_S}(S))|$ *always* yield $\widehat{\alpha} = \alpha$.

7.2.2.3 Case $|\sigma_{p_R}(R) \bowtie_{p_J} \sigma_{p_S}(S)|$

We propose to use Formula (7.5). The estimates are accurate if CAMA holds. We show the accuracy of this approach in the evaluation, in particular in Section 7.3.2.1. To illustrate CAMA, recall the example from the introduction. We

had α=2.5 and $\mathcal{I}$=2, resulting in a true join size of 5. With accurate inputs, we have α_{FK}=$\frac{6}{3}$=2, which yields the estimate $\alpha_{FK}{\cdot}\mathcal{I} = 2{\cdot}2 = 4$, not far off the true join size. For comparison, the independence assumption as in System R [78] yields $s_{B\geq 3}{\cdot}s_{4\leq Z\leq 10}{\cdot}|R{\bowtie}S| = \frac{2}{5}{\cdot}\frac{6}{9}{\cdot}9 = 2.4$, a significant underestimate.

7.2.3 Error Analysis

We present 95% confidence intervals for $\mathcal{D}$ and $\mathcal{I}$ based on their estimates $\widehat{\mathcal{D}}$ and $\widehat{\mathcal{I}}$ from Eq. (7.1) and (7.2) respectively.

We start with the analysis of the error for a single bucket sketch as used for $\widehat{\mathcal{D}}$ subject to predicate p_T for table $T{\in}\{R,S\}$. For the merged HLL from buckets *contained* by p_T, i.e., $H_{\lfloor\cup\rfloor}$, only the error of HLL applies. HLLs estimation errors are normaly distributed [32], which implies that a 95% confidence interval is bounded by $\epsilon_{HLL}{:=}1.96{\cdot}SE_{HLL}$, where $SE_{HLL}{:=}\frac{1.04}{\sqrt{m}}$ is the standard error of an HLL with m counters [32]. It follows that a 95% confidence interval for the true NODV in the *contained buckets* is

$$|H_{\lfloor\cup\rfloor}| \pm \epsilon_{HLL}.$$

Let δ=$|H_{\lceil\cup\rceil}|-|H_{\lfloor\cup\rfloor}|$ denote the estimated NODV in the intersected but not contained buckets. In the worst cases, all or none of the δ distinct values are covered by p_T. The *lower bound estimate* reflects the case that *none* of the δ distinct values is covered by p_T and is given by $|H_{\lfloor\cup\rfloor}|{-}\epsilon_{HLL}$, but at least one, since we assume no query has an empty result. The *upper bound estimate* reflects the case that *all* of the δ distinct values are covered by p_T and is given by $|H_{\lceil\cup\rceil}|{+}\epsilon_{HLL}$.

The regret [43], the (expected) worst case multiplicative error, is minimized by their geometric mean

$$\sqrt{(|H_{\lfloor\cup\rfloor}| - \epsilon_{HLL}) \;\cdot\; (|H_{\lceil\cup\rceil}| + \epsilon_{HLL})}$$

which we approximate by $\widehat{\mathcal{D}}$, computed as in Eq. (7.1). Finally, observe the following regarding the geometric mean of two numbers a, b:

$$\sqrt{a\cdot b}\cdot\Delta = b \iff \Delta = \sqrt{\tfrac{b}{a}} \tag{7.6}$$

and Δ^{-1} satisfies $\sqrt{a\cdot b}\cdot\Delta^{-1} = a$.

With Eq. (7.6), we have with 95% confidence

$$\mathcal{D} \in \left[\widehat{\mathcal{D}}\sqrt{\frac{|H_{\lfloor\cup\rfloor}|}{|H_{\lceil\cup\rceil}|}} - \epsilon_{HLL},\;\; \widehat{\mathcal{D}}\sqrt{\frac{|H_{\lceil\cup\rceil}|}{|H_{\lfloor\cup\rfloor}|}} + \epsilon_{HLL}\right] \tag{7.7}$$

That is, $\mathcal{D}$ deviates from $\widehat{\mathcal{D}}$ by a factor of at most $\sqrt{\frac{|H_{\lceil\cup\rceil}|}{|H_{\lfloor\cup\rfloor}|}}{+}\epsilon_{HLL}$. This factor can be handed to the query optimizer as a reliability measure.

Adding/subtracting ϵ_{HLL} in the above interval bounds is justified by the following inequality, where a, b are HLL estimates of value d:

$$\sqrt{a \cdot b} \cdot \Delta = d \Longleftrightarrow \Delta = \frac{d}{\sqrt{a \cdot b}} \leq \frac{b+\epsilon_{HLL}}{\sqrt{a \cdot b}} \leq \sqrt{\frac{b}{a}} + \epsilon_{HLL},$$

where the first inequality holds true with 95% confidence in the worst case and the second inequality holds true if $\sqrt{a \cdot b} \geq 1$.

We now analyze the error of $\widehat{\mathcal{I}}$ subject to two predicates p_R and p_S using two bucket sketches $\mathcal{B}_R$ and $\mathcal{B}_S$. The rationale is similar as before. However, for HLL intersection using BML we do not know the standard deviation or a similar metric. Note that [68] derive error bounds from a function that approximates the relationship between counters in the HLL sketches, but those bounds do not match our observations. We present 95% *predictive confidence intervals* [16], computed as described in Section 7.3.2.2. The lower and upper error bound of this interval are $\epsilon^-_{BML} \coloneqq -0.75 \cdot k$ and $\epsilon^+_{BML} \coloneqq 0.53 \cdot k$ respectively, where k is the smaller NODV estimate, as obtained from the two merged HLLs to be intersected.

Then, with 95% confidence, the true intersection of the *contained buckets* in $\mathcal{B}_R$ and $\mathcal{B}_S$ lies in

$$\left[\left|H^R_{\lfloor\cup\rfloor} \cap H^S_{\lfloor\cup\rfloor}\right| + \epsilon^-_{BML},\ \left|H^R_{\lfloor\cup\rfloor} \cap H^S_{\lfloor\cup\rfloor}\right| + \epsilon^+_{BML}\right].$$

However, the lower bound is at least one, since we assume no query result is empty. Note that ϵ^-_{BML} is a negative number. Further note that $|H^R_{\lfloor\cup\rfloor} \cap H^S_{\lfloor\cup\rfloor}| + \epsilon^-_{BML}$, but at least one, is also the *lower bound estimate for* $\mathcal{I}$.

The *upper bound estimate* is the intersection size estimate of the merged HLLs from *intersected buckets* plus the upper error bound

$$\left|H^R_{\lceil\cup\rceil} \cap H^S_{\lceil\cup\rceil}\right| + \epsilon^+_{BML}$$

Again, the regret is minimized [43] by the geometric mean

$$\sqrt{\left(|H^R_{\lfloor\cup\rfloor} \cap H^S_{\lfloor\cup\rfloor}| + \epsilon^-_{BML}\right) \cdot \left(|H^R_{\lceil\cup\rceil} \cap H^S_{\lceil\cup\rceil}| + \epsilon^+_{BML}\right)}$$

which we approximate by $\widehat{\mathcal{I}}$, computed as in Eq. (7.2). Hence, for an estimate $\widehat{\mathcal{I}}$, with 95% confidence

$$\mathcal{I} \in \left[\widehat{\mathcal{I}} \cdot \Delta_{\widehat{\mathcal{I}}} + \epsilon^-_{BML},\ \widehat{\mathcal{I}} \cdot \Delta^{-1}_{\widehat{\mathcal{I}}} + \epsilon^+_{BML}\right], \tag{7.8}$$

where $\Delta_{\widehat{\mathcal{I}}} \coloneqq \sqrt{\frac{|H^R_{\lceil\cup\rceil} \cap H^S_{\lceil\cup\rceil}|}{|H^R_{\lfloor\cup\rfloor} \cap H^S_{\lfloor\cup\rfloor}|}}$ follows from Eq. (7.6).

7.2.4 Extensions and Limitations

An extension to multiple predicates on one column, e.g. $A{\leq}c_2 \wedge A{\neq}c_3 \vee A{>}c_3$ is straightforward. Still, the merged HLL sketches $H_{\lfloor\cup\rfloor}$ and $H_{\lceil\cup\rceil}$, corresponding to the contained and intersected buckets, are found via Algorithm 2. All other steps remain unchanged.

An extension to multi-column predicates is feasible for disjunctions, e.g., $A{\geq}\ c_1 \vee B{\leq}c_2$. Note that traditional histograms do not support this. In our method, each predicate is handed to Algorithm 2 separately. Then, the outputs $H^A_{\lfloor\cup\rfloor}$ and $H^B_{\lfloor\cup\rfloor}$ are merged. Similarly $H^A_{\lceil\cup\rceil}$ and $H^B_{\lceil\cup\rceil}$ are merged. All other steps remain unchanged. For the average multiplicity estimate, we recommend α_{Base}.

Conjunctive predicates on different columns require an HLL representation after intersect operations. Zhou et al. [97] present an approach. All other steps are the same as for disjunctions.

Negated predicates can be handled well, e.g., a negated range predicate is equivalent to the disjunction of two ranges.

Bucket sketches are designed for the very common key/foreign-key joins. For many-to-many joins between tables R and S, intersection size estimation $\mathcal{I}$ remains the same. For average multiplicity, one obtains *two* estimates $\widehat{\alpha}^R_{FK}$ and $\widehat{\alpha}^S_{FK}$ via Eq. (7.5), one estimate for each bucket sketch. On average, each join value occurs $\widehat{\alpha}^R_{FK}$ times in R and $\widehat{\alpha}^S_{FK}$ in S. The join size estimate is $\widehat{\mathcal{I}}{\cdot}\widehat{\alpha}^R_{FK}{\cdot}\widehat{\alpha}^S_{FK}$. Implicitly, the estimate for many-to-many joins assumes uniformity of the join columns.

To avoid CAMA, one must explicitly count multiplicities, e.g. via AKMV sketches [17]. To use AKMV sketches, essentially no change has to be made to the bucket sketch, except for the replacement of the sketch. For intersection size estimation, one then uses the AKMV technique, which we presented in Section 6.1. This also addresses the uniformity assumption for many-to-many joins. However, the memory consumption for AKMV sketches is larger.

Bucket sketches are insert-updateable. For a tuple (g, j), find the bucket corresponding to g, insert j into the HLL and increment the *Freq* field. HLLs do not support deletions and, hence, bucket sketches are not delete-updateable. [68] describe how to extend HLLs to support a limited number of deletions.

For a non-numerical partitioning attribute, bucket indexing is adjusted accordingly, e.g., via binary search over bucket boundaries or via dictionary encoding as in [94]. For small domains, one can build one bucket per category/distinct value. The latter simplifies estimation as the buckets *contained* and *intersected* by some predicate are the same.

7.3 Evaluation

In our evaluation, we use the **data sets** (1) TPC-H with a scale factor of 1, (2) data sets generated as described in Section 7.3.1, as well as (3) the two real-world data sets *Instacart* [8], and the *IMDb* [55].

All **queries** in this evaluation follow the structure of the following two types of queries: $\sigma_{p_R}(R) \bowtie_{p_J} S$ and $\sigma_{p_R}(R) \bowtie_{p_J} \sigma_{p_S}(S)$, where p_J is a fixed equality join predicate, and p_R and p_S are range selection predicates, e.g., $c_1 \leq R.B \leq c_2$ for attribute $R.B$. c_1 and c_2 are generated by randomly picking two values from the domain of B. The smaller value is assigned to c_1 and the larger value to c_2. Thus, note that all selection predicates in this evaluation are range predicates. We ran 1,000 queries (at least) for all results we present. Note that we do not consider $R \bowtie_{p_J} \sigma_{p_S}(S)$ queries since their result size can be estimated using single table cardinality estimation.

The q-error, defined as $\max\left(\frac{e}{v}, \frac{v}{e}\right)$ for an estimate e of value v, is an **error metric** that links directly to plan quality [66] and is hence a de facto standard in query optimization. However, to illustrate under- and overestimates, we use the *symmetric relative difference*, defined as $\frac{e-v}{\min(v,e)}$. Taking the absolute of this error and adding one, yields the q-error.

For **our approach** we use bucket sketches with 128 buckets. The HLL of each bucket has 8 counters. A join size estimate is always $\widehat{\alpha} \cdot \widehat{\mathcal{I}}$. The bucket sketches use Microsoft's Binomial Mean Lookup estimator [68], which we know from Section 6.2.2, to estimate intersection sizes. The average multiplicity estimate $\widehat{\alpha}$ depends on the query. In case of $\sigma_{p_R}(R) \bowtie_{p_J} S$, we give the join size estimates using $\widehat{\alpha}_{Base}$ from Formula (7.3) and $\widehat{\alpha}_{CtrFreq}$ from Formula (7.4). In case of $\sigma_{p_R}(R) \bowtie_{p_J} \sigma_{p_S}(S)$, we use $\widehat{\alpha}_{FK}$ from Formula (7.5) as the average multiplicity estimate and give two join size estimates: one where the numerator and denominator in Formula (7.5) are the true values (*Our approach ideal*), and a second where a bucket sketch derives estimates (*Our approach real*).

We use the following estimators for **comparison**: (1) Bernoulli sampling, (2) correlated sampling [86], (3) two-level sampling [23], (4) System R [78], and, (5) Local deep learning [93] trained with 100k queries and their recommended architecture. For the sampling-based estimators, the (expected) sample size is 0.1% the table size, but at least 100 tuples. A description of all these estimators is given in the related work section, Section 4.

7.3.1 Data Generation

For the interested reader, we describe the creation process of our generated data sets. Note that our evaluation additionally uses TPC-H and two real-world data sets.

We generated 36 data sets. Each with key table R and FK table S, defined as in Table 7.1. We always have $|S|{=}3M$. The key table size varies, $\{|R| \in 1k, 10k, 100k, 1M\}$. Note that $|R|$ determines the domain of the FK attribute $S.F$.

The key attribute $R.K$ simply contains the values $1, \ldots, |R|$. The values of $R.B$ are sampled uniformly from $[0, |R|]$. Equivalently, the values of the $S.Z$ are sampled uniformly from $[0, |S|]$. The skewed FK attribute values in $S.F$ are generated as follows: First, each key from $R.K$ is inserted once in the first $|R|$ slots of $S.F$. If $|R|{=}1M$, the remaining $S.F$ values are inserted following a Zipf distribution $Z(\Theta, 1M)$, where we use $\Theta \in \{0, 0.5, 1\}$. Each $v \in \{1, \ldots, |R|\}$ is

added $z_v \cdot (|S| - |R|)$ times to $S.F$, where z_v denotes the relative frequency of v in the aforementioned distribution $Z(\Theta, 1M)$.

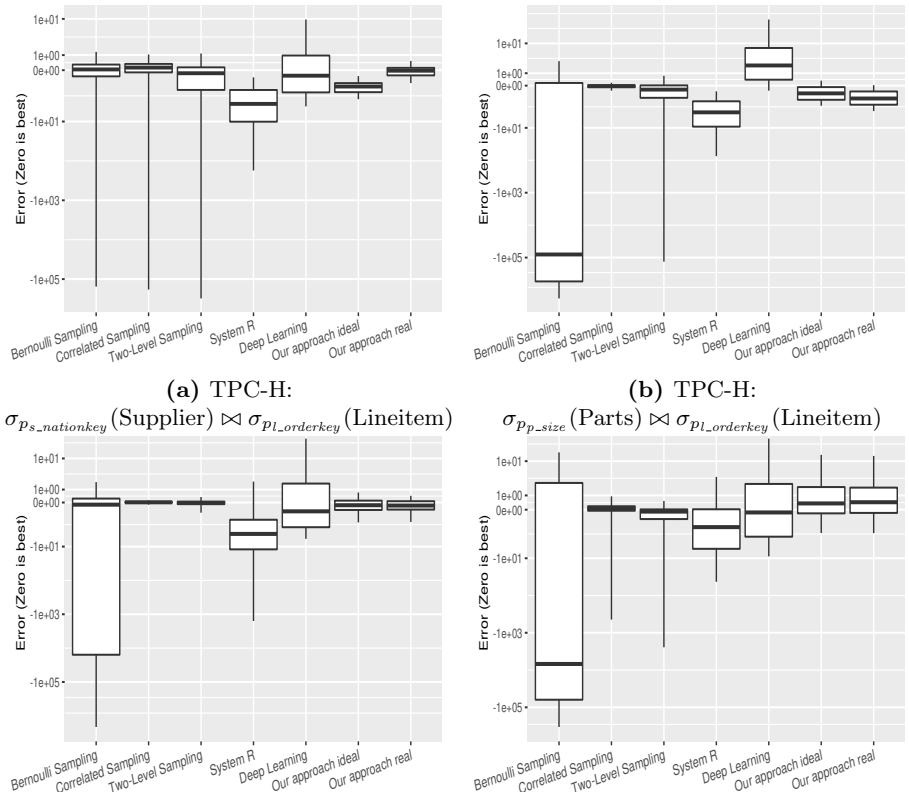
For $|R| \in \{1k, 10k, 100k\}$, instead of using $Z(\Theta, |R|)$, we derive a distribution from $Z(\Theta, 1M)$. The motivation is improved comparability by having Θ have the same effect on the join size for different $|R|$. In particular, for a selection predicate $q \equiv 1 \leq R.K \leq x \cdot |R|$ with $x \in [0, 1]$, we want that $|\sigma_q(R) \bowtie_{p_J} S|$ is the same for all $|R|$ we use. However, this is not the case for Zipf distributions with different domain sizes. Hence, for $|R| \in \{1k, 10k, 100k\}$, the relative frequency of 1 is the sum of the first $\frac{1M}{|R|}$ relative frequencies z_v, the relative frequency of 2 is the sum of the next $\frac{1M}{|R|}$ relative frequencies z_v, and so on.

The relationship between attributes in a set $\mathcal{A}$ is controlled by a correlation parameter ρ. Instead of Pearson correlation coefficient, which measures linear correlation only, our ρ means that all attributes in $\mathcal{A}$ are first sorted, and then an expected fraction of $1-\rho$ in each attribute is randomly shuffled. For ρ, we use 0.2, 0.5, and 0.8.

Combining each $|R| \in \{1k, 10k, 100k, 1M\}$ with each $\Theta \in \{0, 0.5, 1\}$ and each $\rho \in \{0.2, 0.5, 0.8\}$ gives 36 different data sets.

7.3.2 Accuracy Join Size Estimation

Figure 7.3 shows the accuracy of our approach and the competing estimators for the third-party data sets TPC-H, Instacart and IMDb. In addition, tables and 7.2 and 7.3 show the corresponding worst errors, i.e., the most negative and most positive p-error values. In Figure 7.3a, we look at the join $\sigma_{p_{s\_nationkey}}(\text{Supplier}) \bowtie \sigma_{p_{l\_orderkey}}(\text{Lineitem})$ from TPC-H. Lineitem is a fact table with around 6M entries and Supplier a smaller dimension table with 10k entries. Before joining, each relation is filtered. All sampling methods do mostly well. However, as the 95% quantile shows, underestimates by a large factor occur. While sampling is very versatile, a major problem is that one may not find any, or only very few, join tuples after selective predicates were applied to the samples. This is especially reflected in the worst case errors: The most negative/positive errors for Bernoulli sampling are -783825 / 12.27369. For correlated sampling we have -1835033 / 4.41. For two-level sampling they are -1369075 / 8.84. Note that the smallest errors all occur in cases where no sample tuples qualify, in which case the default estimate of 0.5 is returned. This means that there was a query where no sample tuples for correlated sampling qualified but the actual join size was larger than 900k! The System R approach underestimates by a factor of ≥ 10 in more than 25% of the queries. The most negative/positive errors are -186482 / -0.08. The local deep learning estimator avoids heavy underestimates, but tends to overestimate. This is also reflected by the most negative error of -30 and the most positive error of 99.21. Our approach gives good estimates, with the median error around 0 in the idealistic case, and slightly above 0, but below 1, in the realistic case. The most negative/positive errors are -260.84 / 1.38 for the idealistic approach and -114.04 / 24.02 for the realistic approach.



(a) TPC-H: $\sigma_{p_{s\_nationkey}}(\text{Supplier}) \bowtie \sigma_{p_{l\_orderkey}}(\text{Lineitem})$

(b) TPC-H: $\sigma_{p_{p\_size}}(\text{Parts}) \bowtie \sigma_{p_{l\_orderkey}}(\text{Lineitem})$

(c) Instacart: $\sigma_{p_{hour\_of\_day}}(\text{Orders}) \bowtie \sigma_{p_{product\_id}}(\text{Orders\_Products})$

(d) IMDb from Join Order Benchmark: $\sigma_{p_{production\_year}}(\text{Title}) \bowtie \sigma_{p_{company\_id}}(\text{Movie\_Companies})$

Figure 7.3: Errors for different estimators in the third-party data sets. The whiskers show the 5% and 95% percentile.

The results for $\sigma_{p_{p\_size}}(\text{Parts}) \bowtie \sigma_{p_{l\_orderkey}}(\text{Lineitem})$ are shown in Figure 7.3b. Bernoulli Sampling performs badly, with a huge median estimation error, since in many cases no join tuples are found in the samples. The most negative/positive errors for Bernoulli sampling are -4414351 / 42.16. Correlated sampling performs best, but the sample size can be significantly larger than expected, as it must sample *all* tuples with certain join attribute values. The most negative/positive errors are -8683 / 1.25. For the other estimators, the most negative/positive errors are: -827145 / 15.50 for two-level sampling, -91407 / -0.08 for System R, -97/29.03 for local deep learning, -175.50 / 46.65 for our approach ideal, and 1065.27 /28.58 for our approach real.

Figure 7.3c shows the largest join in this evaluation, coming from the real-world Instacart data set: $\sigma_{p_{hour\_of\_day}}(\text{Orders}) \bowtie \sigma_{p_{product\_id}}(\text{Orders\_Products})$. Or-

ders contains around 3M entries, and Orders_Products is a fact table with >32M entries. System R underestimates by a factor of 10 for more than 25% of the queries. The local deep learning estimator appears to be biased towards overestimates. Bernoulli sampling fails in many cases. Correlated and two-level sampling perform great. Our approach has very low errors as well. Also note the the most negative and most positive errors in Table 7.3. Figure 7.3d shows the results for the IMDb data set with query $\sigma_{p_{production\_year}}(\text{Title}) \bowtie \sigma_{p_{company\_id}}(\text{Movie\_Companies})$, the sampling-based estimators suffer from heavy underestimates. Note that Bernoulli sampling underestimates by a factor of $\geq 1'000$ in more than 50% of the queries, two-level sampling still in 5% of the queries. Our approach overestimates, the median error is +2.02 for the idealistic case and +2.08 for the realistic case.

	Supplier ⋈ Lineitem		Parts ⋈ Lineitem	
Estimator	most neg.	most pos.	most neg.	most pos.
Bernoulli Sampling	-783825	12.27	-4414351	42.17
Correlated Sampling	-1835033	4.42	-8683	1.26
Two-Level Sampling	-1369075	8.85	-827145	15.5
System R	-186482	-0.08	-91407	-0.08
Deep Learning	-30	99.21	-97	29.03
Our approach ideal	-260.84	1.38	-175.5	46.65
Our approach real	-114.05	24.02	-1065.27	28.59

Table 7.2: Most negative/positive errors for the TPC-H queries from Figure 7.3.

	Instacart		IMDb	
Estimator	most neg.	most pos.	most neg.	most pos.
Bernoulli Sampling	-10890095	35.05	-1973979	416.21
Correlated Sampling	-3011	2.43	-7821	16.86
Two-Level Sampling	-129121	2.02	-18415	8.66
System R	-2316152	12.18	-11595	20490.92
Deep Learning	-1123	27.65	-137	4028.37
Our approach ideal	-565.16	239.36	-269.81	946.54
Our approach real	-4256.61	221.46	-602.78	1124.2

Table 7.3: Most negative/positive errors for Instacart and IMDb queries from Figure 7.3.

For both Figure 7.3c and Figure 7.3d also note the corresponding worst errors in Table 7.3.

Figure 7.4 and **Figure 7.5** show the errors for queries $\sigma_{p_R}(R) \bowtie_{p_J} \sigma_{p_S}(S)$ over the generated data sets. Recall that the generation of the data sets is described in Section 7.3.1. Both figures show results for data sets with different degrees of skew in the foreign key attribute, and different degrees of correlation among the attributes. In Figure 7.4, the key table R contains only 1,000 tuples. Bernoulli Sampling produces large errors. Correlated sampling and

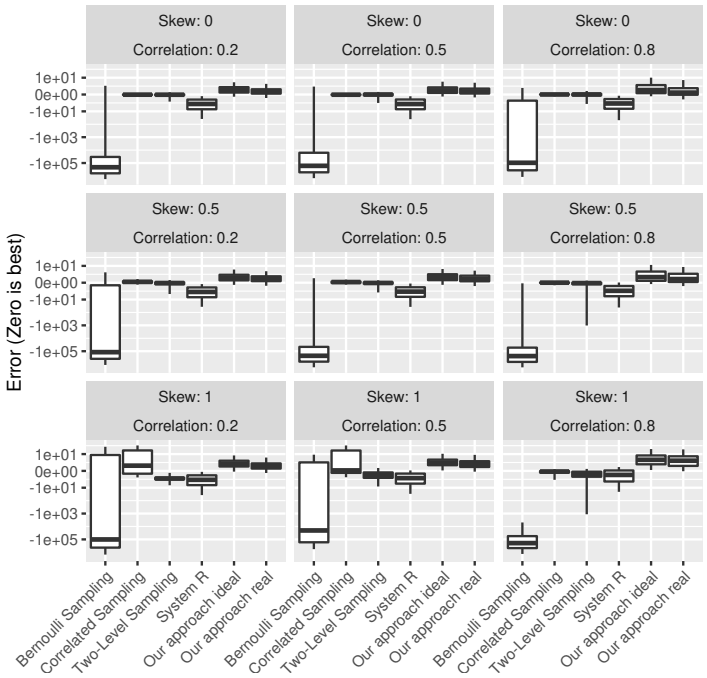


Figure 7.4: Errors for different estimators for generated data. The whiskers show the 5% and 95% percentile. Key table is is 1k. Foreign key table size is 3M.

two-level sampling work great with underestimates slightly more severe than overestimates. Our approach works just as good but with a slight tendency to overestimate. System R performs surprisingly well. Looking at a 100 times bigger key table, Figure 7.5, we see all sampling-based estimators perform significantly worse, especially when skew and correlation is high. Our approach is less vulnerable to large relations, as a bucket sketch is always built on the complete relation.

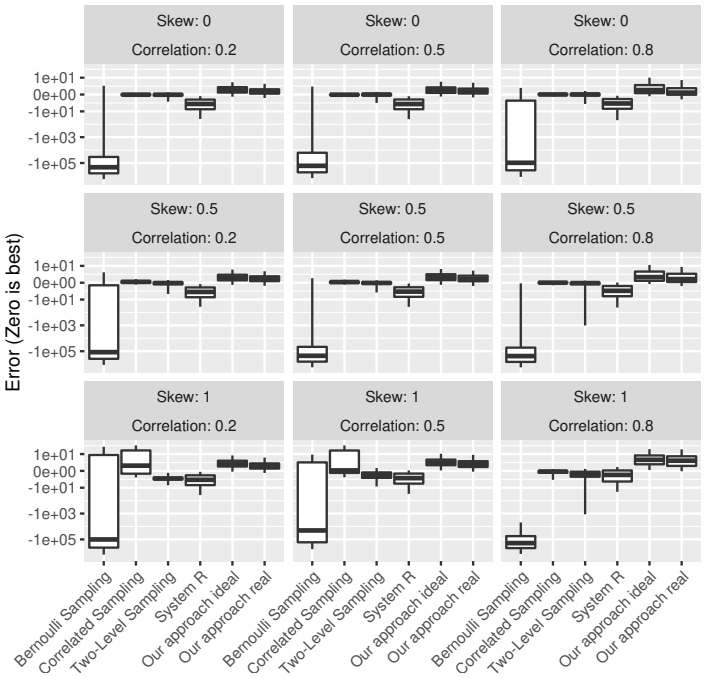
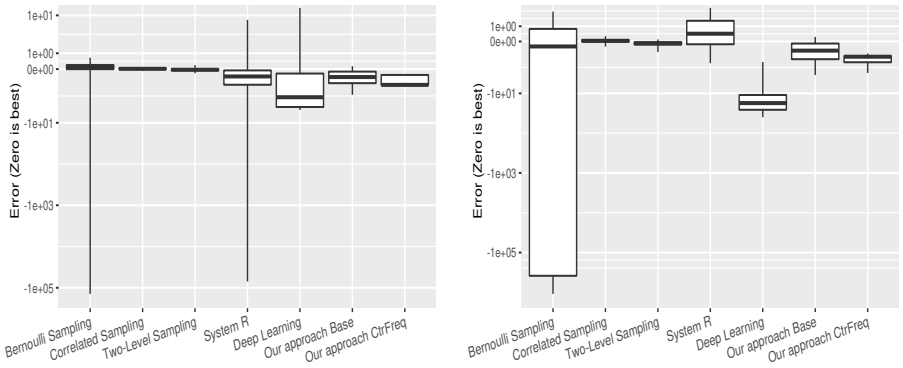


Figure 7.5: Errors for different estimators for generated data. The whiskers show the 5% and 95% percentile. Key table is is 1k. Foreign key table size is 3M.

Figure 7.6 shows two plots for the case when only the key table is filtered. Our approach with $\widehat{\alpha}_{Base}$ is shown as *Our approach Base*, and with $\widehat{\alpha}_{CtrFreq}$ as *Our approach CtrFreq*. Again the deep learning estimator seems to struggle with both real-world data sets, Instacart and IMDb. All other estimators give good estimates for most of the queries. However, Bernoulli sampling and System R have large estimation errors for 5% of the queries. Comparing the errors of System R and our approach with $\widehat{\alpha}_{Base}$, the results indicate that CAMA is more realistic than the independence assumption.



(a) Instacart:
$\sigma_{p_{hour\_of\_day}}(\text{Orders}) \bowtie \text{Orders\_Products}$

(b) IMDb from Join Order Benchmark:
$\sigma_{p_{production\_year}}(\text{Title}) \bowtie \text{Movie\_Companies}$

Figure 7.6: Errors for different estimators in the third-party data sets. The whiskers show the 5% and 95% percentile.

	Instacart		IMDb	
Estimator	most neg.	most pos.	most neg.	most pos.
Bernoulli Sampling	-5621523	18.49	-3485091	25.8
Correlated Sampling	-0.31	0.41	-909	1.77
Two-Level Sampling	-8.05	1.63	-28213	1.34
System R	-2738581	15.21	-17978	12512.8
Deep Learning	-712	- 1.91	-47	16814.19
Our approach Base	-1.28	1.86	-108.42	3.76
Our approach Ctr	-1.00	-0.33	-211.07	1.54

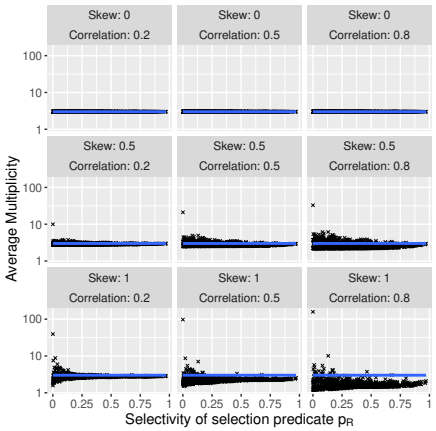
Table 7.4: Most negative/positive errors for the boxplots from Figure 7.6.

7.3.2.1 Accuracy Average Multiplicity Estimation

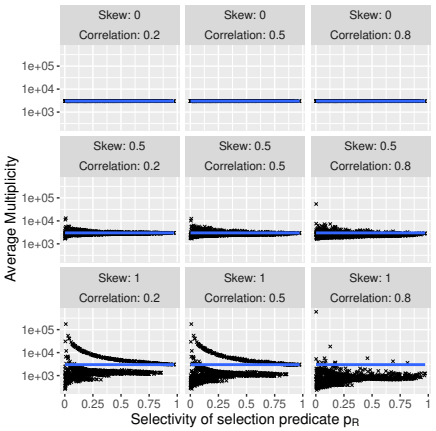
This section analyses the accuracy of average multiplicity estimation.

We begin with the estimator $\widehat{\alpha}_{Base}$ for queries of the form $|\sigma_{p_R}(R) \bowtie_{R.K=S.F} S|$. For different degrees of correlation and skew in the generated data sets, **Figure 7.7** illustrates how well this formula predicts the average multiplicity in the selectivity of p_R. In the plot, each black cross denotes the average multiplicity α as observed after applying one selection predicate p_R with selectivity as indicated by the x-axis. The blue line denotes the estimated average multiplicity $\widehat{\alpha}_{Base}$. The first row of the plots shows that correlation does not affect the quality of the estimate if there is no skew. In that case, we have $\widehat{\alpha}_{Base} = \alpha$. The reason is that it does not matter which primary keys are selected, since all $R.K$ values match an equal number of FK values in $S.F$.

Similarly, as shown in the lower left corner, high skew has a low impact on the quality of the estimate if the correlation between the join attribute $R.K$ and the selection attribute is low, unless the selectivity of p_R is close to zero. Low correlation means that those values in $R.K$ with many matching foreign keys



(a) R table size: 1M;
α before selection p_R: 3



(b) R table size: 1,000;
α before selection p_R: 3,000

Figure 7.7: Each plot compares observed average multiplicities α (black crosses) with the estimate $\widehat{\alpha}_{Base}$(blue line) as by Formula (7.3). Each plot refers to a different degree of skew in the foreign keys S.F and a different degree of correlation between the join predicate $R.K$ and the selection predicate $R.B$.

in $S.F$ are independent of the attribute values to which p_R is applied. Hence, any selection, as chosen by p_R, yields a similar average multiplicity. Selectivities close to zero impose a problem. Here it might be that only a single tuple of R is selected and its multiplicity with $S.F$ may be either very large or low. As the selectivity of p_R approaches 1, the predicate loses its effect and the observed average multiplicities converge to the estimate $\widehat{\alpha}_{Base}$.

The estimates become inaccurate for highly skewed values in $S.F$ and high correlation between $R.K$ and the selection attribute in p_R, as shown in the plots in the lower right corner of Figure 7.7. In addition, in these cases the estimates worsen as the size of R is reduced from one million to one thousand, cf. Figures 7.7a and 7.7b, since in a small relation a value in $R.K$ finds, on average, much more matches in $S.F$.

As expected, the estimates can be improved with the estimator $\widehat{\alpha}_{CtrFreq}$ from page 106, cf. **Figure 7.8**. Each boxplot shows the p-error distribution for one correlation and skew combination. Note that the estimates are very accurate.

For queries of the form $|R \bowtie_{p_J} \sigma_{p_S}(S)|$ and $|\sigma_{p_R}(R) \bowtie_{p_J} \sigma_{p_S}(S)|$, we have presented the estimator $\widehat{\alpha}_{FK}$. Recall that, for queries of the form $|R \bowtie_{p_J} \sigma_{p_S}(S)|$, perfect estimates for $|\sigma_{p_S}(S)|$ and $|\pi^D_Y(\sigma_{p_S}(S))|$ result in $\widehat{\alpha}_{FK} = \alpha$. Indeed this can be empirically observed, cf. **Figure 7.9a**. For comparison, cf. **Figure 7.9b**, we show that an estimate that relies only on the selectivity of p_S gives good estimates only if the data is not too skewed and correlated. The two blue lines each show a fitted curve through the observed α values. For no skew and low correlation, there clearly exists a mapping between the selectivity of p_S

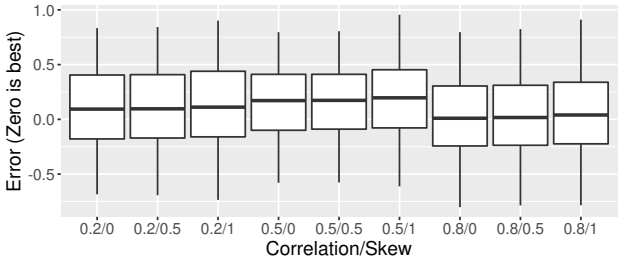
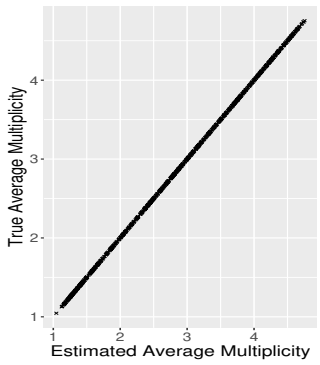
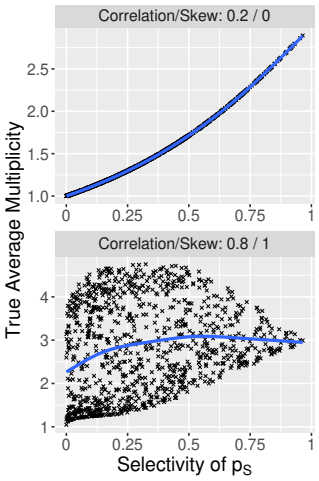


Figure 7.8: Distribution of errors between true and estimated average multiplicity as by Formula (7.4). Each box plot corresponds to one skew and correlation combination from Figure 7.7 and is computed over 1,000 queries. The whiskers show the 5% and 95% percentile.



(a) Scatter plot for correlation = 0.8 and skew = 1. Note that, for $|R \bowtie_{p_J} \sigma_{p_S}(S)|$, we observe $\widehat{\alpha}_{FK} = \alpha$ for accurate inputs to Formula (7.5). For all other correlation and skew combinations the plot looks the same.

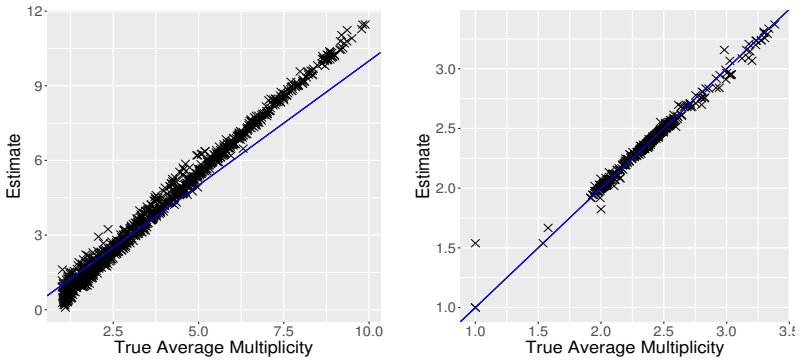


(b) α is not a function of the selectivity of p_S if correlation and skew is high.

Figure 7.9: R table size is 1M. S table size is 3M. Average multiplicity before p_S is 3.

and α. However, for high skew and high correlation, one cannot predict α from the selectivity of p_S.

Finally, we confirm that the above observations for $\widehat{\alpha}_{FK}$ and $\widehat{\alpha}_{CtrFreq}$ hold in the real-world data sets Instacart and IMDb. **Figure 7.10** shows the errors for average multiplicity estimation. Figure 7.10a refers to the queries from Figure 7.3c and shows a scatter plot with observed average multiplicities on the x-axis and estimates as produced by Formula 7.5, $\widehat{\alpha}_{FK}$, on the y-axis. Points above the diagonal blue line represent overestimates, points below represent underestimates. The most negative/positive error is −11.40/0.59. Figure 7.10b refers to the queries $\sigma_{p_{production\_year}}(\text{Title}) \bowtie \text{Movie\_Companies}$. The estimates on



(a) Estimate via Formula 7.5 ($\widehat{\alpha}_{FK}$) **(b)** Via Formula 7.4 ($\widehat{\alpha}_{CtrFreq}$)

Figure 7.10: Scatter plot for observed average multiplicities (x-axis) and estimates (y-axis).

the y-axis are produced by Formula 7.4, i.e., $\widehat{\alpha}_{CtrFreq}$. The most negative/positive error is $-0.09/0.53$. In addition, note that most observations are clustered around the naive estimate $\widehat{\alpha}_{Base}$=2.4.

7.3.2.2 Confidence interval for BML

In this section, we present a 95% *predictive confidence interval* [16] for HLL intersection using BML. We use the *naive method* and the *asymmetric extension* described in their appendix [16].

For sets K and F, we generated all combinations of $|K|, |F| \in \{50, 10^2, 10^3, 10^4, 10^5\}$ and $|K \cap F| \in \{.001, .005, .01, .02, .1, .5, .7, .9\} \cdot \min(|K|, |F|)$. We consider only cases where $|K \cap F| \geq 10$. For each combination, we built 1,000 HLL pairs HLL_K, HLL_F, each pair with a different hash function.

Let $\epsilon_{BML} := |HLL_K \cap HLL_F| - |K \cap F|$ be the *signed residual* [16]. Further, let $k = \min(|K|, |F|) \approx \min(|HLL_K|, |HLL_F|)$ be the largest possible intersection size. In Table 7.5 we report the lower bound $\epsilon(k)^-_{BML}$ and upper bound $\epsilon(k)^+_{BML}$ of the 95% predictive confidence interval [16] for different k. Clearly, the width of the interval depends on k. However, dividing the bounds by k yields normalized interval bounds that remain approximately stable for different k. Hence, we average and set $\epsilon^-_{BML} := -0.75 \cdot k$ and $\epsilon^+_{BML} := 0.53 \cdot k$. Finally, the 95% predictive confidence interval is $|K \cap F| \in |HLL_K \cap HLL_F| + \begin{cases} \epsilon^-_{BML} \\ \epsilon^+_{BML} \end{cases}$.

k	$\epsilon(k)^-_{BML}$	$\epsilon(k)^+_{BML}$	$\frac{\epsilon(k)^-_{BML}}{k}$	$\frac{\epsilon(k)^+_{BML}}{k}$
50	-29	25	-0.58	0.50
100	-70	53	-0.70	0.53
1000	-804	579	-0.80	0.57
10,000	-8455	5429	-0.84	0.54
100'000	-81524	53483	-0.81	0.53

Table 7.5: Predictive 95% confidence interval of HLL intersection

$\|R\|$	Bernoulli	Correlated	Two-level	Learn.	Bsketch
1k	24.8	24.8 - 372.7	7.2 - 8.2	≈ 1700	3.6
10k	24.8	24.8 - 375.2	7.2 - 10.8	≈ 1700	3.6
100k	24.8	24.8 - 364.7	7.3 - 11.8	≈ 1700	3.6
1M	32.0	32.0 - 246.0	10.4 - 15.7	≈ 1700	3.6

Table 7.6: Memory consumption in kilobytes. $|R|$ shows the number of rows in the key table. The FK table has 3M rows in all cases. Our approach, *Bsketch*, is listed last.

7.3.3 Memory Consumption

Table 7.6 shows the memory consumption in kilobytes for the different approaches. System R is not listed due to its negligible memory costs. For each estimator, we give the range of observed memory footprints when applied to the generated data sets. The expected size of the sample of a relation T is $\max(0.1\%|T|, 100)$. Each tuple has size 8 bytes: two integers or one pointer. The memory consumption of Bernoulli sampling is always predictable. For correlated sampling and two-level sampling, the actual sample size depends on skew. For two-level sampling the memory consumption stays within tight bounds. The memory consumption of correlated sampling can vary by a large factor. For local deep learning, we list the size of the materialized model, which has a size of around 1.7MB.

We use two bucket sketches with 128 buckets each. Each bucket contains one 4 byte *Freq* field and one HLL sketch with 8 counters, each of size 1 byte. Additionally, the buckets of key table R each contain one *CtrFreq* field. Hence, our approach consumes $2{\cdot}128{\cdot}(4B{+}8B){+}128{\cdot}4B{=}3584B$. Sketches summarize a lot of data with very little memory. For this reason, our approach has a small memory footprint. Consider this: With 3584 bytes you could only sample 448 8-byte tuples.

7.3.4 Time Consumption

The construction time for a bucket sketch is in $O(n)$. Single-threaded construction takes 13, 85, 815, and 8'196 microseconds for tables with 1k, 10k, 100k, and 1M entries, respectively. To further improve construction time, multiple bucket sketches can be constructed in parallel and then merged into one bucket sketch.

Key table	FK table	Inc-Exc	BML	Ertl
1'000	3'000'000	3	31	20
10'000	3'000'000	3	40	38
100'000	3'000'000	4	47	26
1'000'000	3'000'000	4	56	55

Table 7.7: Microseconds to Estimate $\mathcal{I}$ from two Bucket Sketches.

n	Size	|CtrFreqHT|*	Mean abs(Error)	95% abs(Error)
8	0.22 KB	4 KB	21.56	72.67
16	0.45 KB	4 KB	11.76	45.07
32	0.90 KB	4 KB	6.93	20.87
64	1.79 KB	4 KB	3.17	9.21
128	3.58 KB	4 KB	2.25	5.47
256	7.17 KB	4 KB	1.68	4.53
512	14.34 KB	4 KB	1.51	4.22

*temporary, only during construction

Table 7.8: Space-Accuracy Tradeoff

Table 7.7 shows the average time it takes to estimate an intersection size from two bucket sketches in microseconds. The fastest approach is to use inclusion-exclusion principle (Inc-Exc) as the underlying estimation method. Most time is consumed by the bucket sketches, which must handle the selection predicates and output a merged HLL sketch, as described in Section 7.2.1. Binomial mean lookup by Microsoft (BML) and Ertl's approach (Ertl) have similar estimation times. For both approaches, the estimation time depends on the size of the key table, which determines the possible intersection size, i.e, the search space for an estimate. For comparison, the deep learning approach averages around 10'000μs=10ms, where we used TensorFlow's C API.

7.3.5 Space-Accuracy Tradeoff

Table 7.8 shows, for different numbers of buckets, the memory footprint as well as the absolute mean error and absolute 95% error quantile as observed over the generated data sets and queries from Figure 7.4. *Size* denotes the size in KB for two bucket sketches with n buckets each and with the optional *CtrFreq* field in the bucket sketch of the FK table. Note that *Size* is the memory consumption for storing a bucket sketch pair *after* construction. The optional *CtrFreq* field requires a temporary hash table *CtrFreqHT* (see Algorithm 1 line 2) *during* construction. *CtrFreqHT* holds one integer for each distinct join attribute value, which results in $|CtrFreqHT| = 4$KB for Figure 7.4. For comparison, for the join between Supplier and Lineitem from TPC-H $|CtrFreqHT|$ would be 40KB. Note that the 95% error quantile levels out at $n = 128$, which motivates this choice in our evaluation.

7.3.6 Equi Width vs. other Partitioning Schemes

We discussed the structure of bucket sketches in Section 7.2.1.1, where we defined that bucket sketches partition a *partitioning attribute* via equi width. Recall that, unlike frequency histograms, the bucket of a partition does not count the number of tuples that fall into that partition. Instead, the number of distinct values in the *HLL attribute* is counted, where the HLL attribute is generally different from the partitioning attribute. Also note that a proper partitioning scheme requires that all HLL attributes with the same partitioning attribute value fall into the same bucket.

In this section, we analyze the accuracy of bucket sketches under different partitioning schemes. In particular, we compare equi width (EqW) to the two partitioning schemes NoDv0 and NoDv1. NoDv0 and NoDv1 were first defined in a thesis by a student at the University of Mannheim [77]. The motivation for these partitioning schemes is to improve estimates for $|\pi^D_J((\sigma_{p_T}(T)))|$, where J is the HLL attribute. NoDv0 and NoDv1 are inspired by equi depth partitioning from histograms [72]. Recall that, in histograms, equi depth induces buckets with an equal number of tuples. In the following, we give the definition of the three bucketing schemes:

- **EqW:** The partitioning attribute is divided into n buckets of equal width. The time complexity to construct a bucket sketch is in $O(|R|)$, for some relation R.

- **NoDv0:** First, sort a table based on the partitioning attribute. Then, to construct a bucket sketch, iterate over the table and expand the current bucket b by all tuples with the next distinct partitioning attribute value as long as the (approximated) number of distinct HLL attributes values in b does not exceed a threshold θ. The total number of buckets in the bucket sketch is known only after construction. The time complexity to construct a bucket sketch is in $O(|R| \log(|R|))$, for some relation R.

- **NoDv1:** Same as NoDv0, except for the following difference. Denote by V all tuples with the next distinct partitioning attribute value. If adding all tuples V increased the (approximated) number of HLL attributes in the current bucket beyond $2 \cdot \theta$, then the tuples V are placed in a separate bucket.

Recall Section 7.2.1.3, where we have seen how to use bucket sketches for NODV estimation in filtered relations. In the following, we compare the accuracy of an estimate for $|\pi^D_J((\sigma_{p_T}(T)))|$ under the above bucketing schemes. To this end, we repeat an experiment from [77], in which the HLL of each bucket has 64 counters: Let $\mathcal{T}$ be a set of relations, defined as described below. Each relation $T \in \mathcal{T}$ has partitioning attribute G and HLL attribute J, and $|T| = 1$ million. For each partitioning scheme, we construct histograms with $\theta \in \{50, 100, 200, 500, 1000\}$, where EqW is given as many buckets as NoDv1 requires. Then, for 10,000 randomly generated range predicates over $T.G$, we

obtain an estimate for the NODV in J for each constructed histogram. We report the estimation errors over all queries and over all relations in Table 7.9.

Looking at Table 7.9, we note that (1) EqW has the lowest average error (2) NoDv0 and NoDv1 have lower maximum errors than EqW, except for the case $\theta = 1000$, and (3) there is no observable difference between NoDv0 and NoDv1. Note that for queries where either the NODV or its estimate is larger than 4θ, NoDv1 leads to more accurate estimates than EqW and NoDv0 in [77]. Further note that, given NoDv1 partitioning, [77] proved q-θ-acceptability guarantees for the estimates of $|\pi_J^D((\sigma_{p_T}(T)))|$. As of now, it remains open if these guarantees can be carried over to intersection size estimation.

type	theta	avg(q-error)	max(q-error)
EqW	50	1.08	110.67
EqW	100	1.19	124.76
EqW	200	1.32	150.60
EqW	500	1.81	199.63
EqW	1000	3.36	118.83
NoDv0	50	1.16	49.37
NoDv0	100	1.29	58.15
NoDv0	200	1.58	81.13
NoDv0	500	2.38	95.21
NoDv0	1000	3.36	118.83
NoDv1	50	1.16	49.37
NoDv1	100	1.29	58.15
NoDv1	200	1.58	81.13
NoDv1	500	2.38	95.21
NoDv1	1000	3.36	118.83

Table 7.9: Average and maximum q-errors under different partitioning schemes.

We take a closer look at the errors in Table 7.10, where we report the fraction of estimates whose q-errors fall into some range. Again, for $\theta = 1000$ all partitioning schemes lead to the same errors. EqW is the partitioning scheme for which most estimates have a q-error ≤ 2. Partitioning via NoDv0 and NoDv1 again leads to indistinguishable estimation errors. Notably, only for EqW errors larger 100 are observed and in some cases EqW has the highest fraction of large estimation errors (≥ 10). For instance, for $\theta = 100$, 0.28% of EqW estimates have an error ≥ 10, while only 0.19% of NoDv0/NoDv1 estimates have such high errors. However, for some choices of theta, e.g., $\theta = 200$, this pattern is reversed. Only 0.49% of EqW estimates have an error ≥ 10, while 0.68% of NoDv0/NoDv1 estimates have such high errors. Favoring simplicity, we conclude that EqW is a good partitioning scheme for the bucket sketch.

For the interested reader, we detail on $\mathcal{T}$, the set of relations used in the above experiment. For each combination of the following parameters, $\mathcal{T}$ contains three randomly generated tables: For each column G and J, the attribute value ranges are $[0,\ T/x]$, where $x \in \{1, 5, 25, 100, 500\}$. To simulate correla-

type	theta	$q \leq 2$	$2 \leq q \leq 5$	$5 \leq q \leq 10$	$10 \leq q \leq 100$	$q \geq 100$
EqW	50	99.25	0.55	0.11	0.09	0
EqW	100	98.26	1.2	0.27	0.28	0
EqW	200	96.89	2.15	0.47	0.49	0
EqW	500	80.63	16.1	2.14	1.15	0.01
EqW	1000	54.83	27.83	11.28	6.14	0
NoDv0	50	96.93	2.57	0.41	0.09	0
NoDv0	100	93.17	5.88	0.77	0.19	0
NoDv0	200	87.21	9.75	2.37	0.68	0
NoDv0	500	72.83	18.18	5.93	3.08	0
NoDv0	1000	54.83	27.83	11.28	6.14	0
NoDv1	50	96.93	2.57	0.41	0.09	0
NoDv1	100	93.17	5.88	0.77	0.19	0
NoDv1	200	87.21	9.75	2.37	0.68	0
NoDv1	500	72.83	18.18	5.93	3.08	0
NoDv1	1000	54.83	27.83	11.28	6.14	0

Table 7.10: Fraction of q-errors in different error ranges for different partitioning schemes.

tion among attributes, columns are either sorted or not sorted. To introduce skew, the attribute values of both G and J are generated in $[0, 1]$, and then scaled up, via an exponential probability distribution function with parameter $\lambda \in \{0.1, 0.2, 1, 2, 5\}$. To avoid that only small values occur as heavy hitters, which is a property of the exponential distribution, we shift each generated value v_i of attribute $X \in \{G, J\}$ by setting $v_i = (v_i + \frac{z}{2}) \mod z$, where $z = |\pi^D_X(T)|$. This shift applies to either no column or only G or only J (but not both).

7.3.7 Intersection Size Estimation Subject to Predicates

In Section 6, we have introduced and analyzed different intersection size estimation techniques. In this section, we reinvestigate intersection size estimation to verify that the earlier results still hold when the input relations to the intersection are subject to selection predicates.

7.3.7.1 Counters per HLL

First, we compare the three intersection size estimators discussed in Section 6.2.2 using join queries subject to selection predicates over our generated data sets, where we vary the key table size between 1k, 10k, 100k, and 1M entries. **Table 7.11** shows several error quantiles for Microsoft's BML [68] and Ertl's approach [29] for HLLs with 8 and 64 counters. Note that the 1% quantile shows the smallest error of the 1% largest underestimates, and the 99% quantile the smallest error of the 1% largest overestimates. The largest estimation errors occur for large key table sizes. Especially Ertl's approach tends to underestimate as the range of possible intersection sizes increases. However, the estimates significantly improve as we increase the number of counters from 8

		Binomial Mean Lookup						
Size	Counters	1%	10%	25%	50%	75%	90%	99%
1k	8	-18.95	-0.43	-0.12	0.12	0.41	0.73	1.45
1k	64	-63.00	-7.41	0.00	0.23	0.44	0.66	0.97
10k	8	-45.98	-0.74	-0.42	-0.11	0.17	0.47	2.82
10k	64	-44.98	-0.47	-0.21	0.00	0.17	0.34	2.67
100k	8	-41.39	-0.61	-0.07	0.28	0.75	1.44	10.61
100k	64	-41.37	-0.15	0.12	0.36	0.75	1.29	11.25
1M	8	-70.02	0.07	0.44	1.06	2.27	4.57	25.01
1M	64	-67.23	0.25	0.51	1.04	2.20	4.62	24.60
		Ertl's Approach						
Size	Counters	1%	10%	25%	50%	75%	90%	99%
1k	8	-17.90	-0.23	0.02	0.27	0.61	1.00	1.86
1k	64	-17.69	-0.10	0.08	0.25	0.46	0.69	0.98
10k	8	-49.41	-0.58	-0.29	-0.00	0.30	0.63	2.95
10k	64	-45.19	-0.46	-0.21	0.01	0.17	0.33	2.32
100k	8	-24483	-125.38	-0.26	0.22	0.67	1.27	3.45
100k	64	-8029	-24.82	-0.14	0.15	0.37	0.62	1.29
1M	8	-122222	-10860	-0.28	0.57	1.25	2.21	8.30
1M	64	-43381	-165	0.05	0.37	0.61	0.87	3.16

Table 7.11: Binomial Mean Lookup & Ertl's Approach: Error quantiles. 1% shows the largest underestimates. 99% shows the largest overestimates. *Size* is the size of the key table, which is also the maximal possible intersection size.

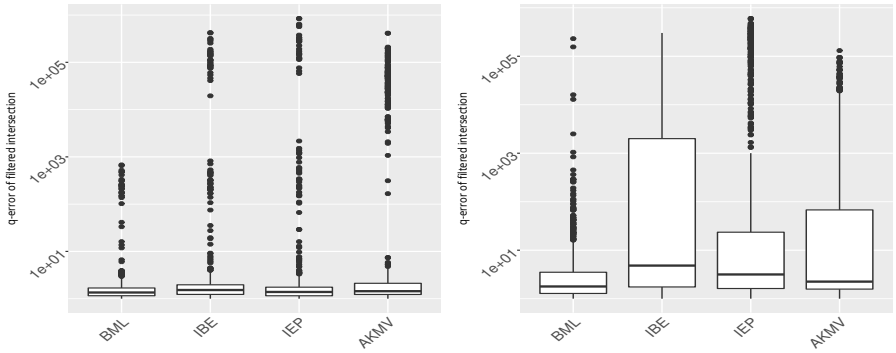
to 64. Looking at Microsoft's BML, note the much lower worst case underestimates. Furthermore, the estimation errors are mostly the same, for HLL sketches with 8 counters and 64 counters. The reason is that their mapping from intersection size to the relationship between the counters of two sketches is independent of the size of the sketches.

For the inclusion-exclusion principle, we only present some numbers. With an median error around 0 for all key table sizes, the estimator is practically median unbiased. However the 1% largest underestimation errors are -18.94, -46.69, -43e3, and -19e4 for key table size 1k, 10k, 100k and 1M.

7.3.7.2 Comparison with AKMV

We rerun the queries from Section 7.3.2 for the Instacart and IMDb dataset. This time, instead of the join size, we want to estimate the intersection size. We consider all three HLL estimators from Section 6.2.2 when used with 8 counters in a bucket sketch of 128 buckets. In addition, we revisit the AKMV intersection size estimator from Section 6.1. To give AKMV a realistic chance, we give the AKMV of each bucket space for 32 hash values (16 times the size of an HLL).

Figure 7.11 shows the results when both, the key and the foreign key table are subject to selection predicates. **Figure 7.12** and **Figure 7.13** show the results when only the key table and the foreign key table, respectively,

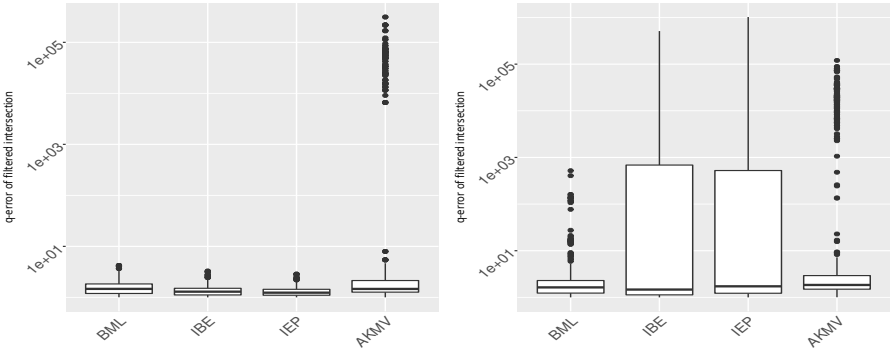


(a) Instacart:
$\pi_{\text{order\_id}}(\sigma_{p_{hour\_of\_day}}(\text{Orders}))\cap$
$\pi_{\text{order\_id}}(\sigma_{p_{product\_id}}(\text{Orders\_Products}))$

(b) IMDb:
$\pi_{\text{id}}(\sigma_{p_{production\_year}}(\text{Title}))\cap$
$\pi_{\text{movie\_id}}(\sigma_{p_{company\_id}}(\text{Movie\_Companies}))$

Figure 7.11: Pred on Both

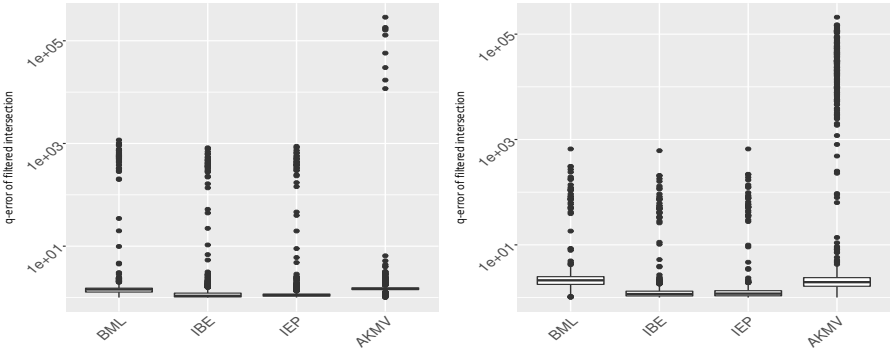
are filtered. In the tables, BML denotes Binomial Mean Lookup, IBE denotes Ertl's approach, and IEP denotes inclusion-exclusion principle. There is no clear winner. BML performs well in all cases. In particular, in no case BML is the worst estimator. In many cases, cf. Tables 7.12 and 7.13, AKMV is the worst estimator. However, in Figure 7.12b, q-errors > 10 occur more frequently for estimates of IBE and IEP than for AKMV. The results support BML as a good estimator for intersection size estimation, also subject to selection predicates.



(a) Instacart:
$\pi_{\text{order\_id}}(\sigma_{p_{hour\_of\_day}}(\text{Orders}))\cap$
$\pi_{\text{order\_id}}(\text{Orders\_Products})$

(b) IMDb:
$\pi_{\text{id}}(\sigma_{p_{production\_year}}(\text{Title}))\cap$
$\pi_{\text{movie\_id}}(\text{Movie\_Companies})$

Figure 7.12: Pred only on Key Table



(a) Instacart:
$\pi_{\text{order\_id}}(\text{Orders})\cap$
$\pi_{\text{order\_id}}(\sigma_{p_{product\_id}}(\text{Orders\_Products}))$

(b) IMDb:
$\pi_{\text{id}}(\text{Title})\cap$
$\pi_{\text{movie\_id}}(\sigma_{p_{company\_id}}(\text{Movie\_Companies}))$

Figure 7.13: Pred only on Foreign Key Table

7.4 Summary

We presented a technique to estimate the size of a key/foreign-key join of two filtered relations. Our technique builds on a model that was developed decades ago [85] and has gained only little attention since. For deriving a join size estimate, we presented a memory-efficient data structure, the bucket sketch, which we use to estimate the intersection size of two data columns after a selection. In addition, we presented formulas to estimate the average multiplicity of a join. Our evaluation takes several state-of-the-art sampling-based join size estimators into account, yet the performance of our approach is very competitive. In particular, in our experiments, our estimator is less prone to heavy underestimates.

Chapter 8

Multiple Joins

8.1 Introduction

In this chapter, the **problem we solve** is to estimate the result cardinality of queries containing multiple joins. Consider, for instance, a star query in which the two dimension tables `D1, D2` join with the fact table `F`:

```
SELECT count(*)
FROM D1 JOIN F ON D1.A = F.A JOIN D2 ON D2.B = F.B
WHERE D1.X < 5 AND D2.Y != "foo";
```

Our approach to derive estimates relies on two data structures: (1) Well-established AKMV sketches [18], which we enhance with new operations, including operations for two-way joins. As well as (2) the *translation grid*, a novel data structure that helps to estimate the size of multi-way joins. To motivate translation grids, observe that in the above example query, the joint frequency distribution between `F.A` and `F.B` influences the size of the three-way join. Here, translation grids come into play. Translation grids allow to determine attribute value pairs that exist, with non-zero frequency, in a joint frequency distribution. We refer to this existence criterion as *joint existence distribution* (JED). However, instead of operating on actual attribute values, translation grids operate on hashes. In particular, translation grids track pairs of hashes that exist, or might exist, in a pair of AKMV sketches.

Figure 8.1 gives an overview of our approach for the above example query. Before processing queries, the translation grid and AKMVs are built for the fact table F. This is illustrated by the dashed arrows pointing from F to the translation grid and the AKMVs $\mathcal{S}_{F.A}$, $\mathcal{S}_{F.B}$. After that, to estimate the result cardinality of the example query, the small dimension tables $D1$ and $D2$ are first filtered and then used to construct AKMVs $\mathcal{S}_{D1.A}$ and $\mathcal{S}_{D2.B}$, respectively[1]. Then, two-way joins are handled by combining individual AKMV sketches into

[1]Alternatively, bucket sketches can be used, where buckets contain AKMVs instead of HLLs.

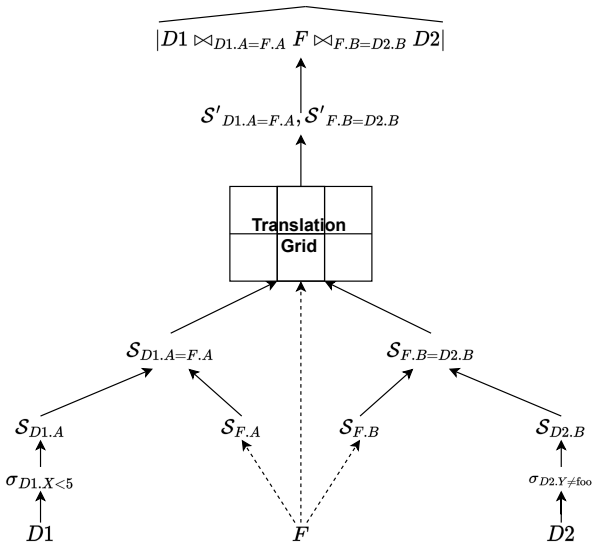


Figure 8.1: High level overview of our approach.

$\mathcal{S}_{D1.A=F.A}$ and $\mathcal{S}_{F.B=D2.B}$. Next, a translation grid processes the combined AKMV sketches $\mathcal{S}_{D1.A=F.A}$, $\mathcal{S}_{F.B=D2.B}$. Finally, based on the output of the translation grid, i.e., two new AKMVs $\mathcal{S}'_{D1.A=F.A}$, $\mathcal{S}'_{F.B=D2.B}$, the query result cardinality is estimated.

The remainder of this chapter is structured as follows: Section 8.2 presents new operations for AKMV sketches. In Section 8.3, we present the translation grid data structure. Then, we evaluate our approach in Section 8.4. Finally, Section 8.5 summarizes this chapter.

8.2 New Operations for AKMV Sketches

This section extends AKMV sketches by two new estimators and one new operation.

As in Section 6.1, where AKMV sketches have been introduced, this chapter makes use of relations R and T with attribute sets C and D, respectively. As earlier, for an AKMV built on attribute $R.C$, k denotes the number of tracked hashes and we always assume the common case $k < d$, where $d := |R.C|_d := |\pi^D_C(R)|$. For simplicity, we use only one deterministic global hash function $H : \circ \to [0, 1]$ that is capable of hashing any input to $[0, 1]$. As in [17], all lemmata assume that H is collision-free. Unless explicitly stated otherwise, all KMVs/AKMVs share the same fixed parameter k.

For an AKMV $\mathcal{S}_E$ constructed on some expression E, we define its *distinct ratio* as the fraction of tracked hashes, i.e. $k/|E|_d$. We say that an AKMV with distinct ratio x is an x-AKMV. In Section 8.3.4, we apply the distinct ratio as a scaling factor. The below lemma shows how to estimate the distinct ratio of $\mathcal{S}_E$.

Lemma 8.2.1. *Recall that, for some AKMV $\mathcal{S}_E$, we denote by $p_E := |\{h_i \in \mathcal{H}_\mathcal{E} : \eta(h_i) > 0\}|$ the number of tracked hashes with multiplicity greater zero. An estimate for the distinct ratio of $\mathcal{S}_E$ is*

$$\frac{p_E}{\widehat{d}_{AKMV}}.$$

Proof. We distinguish two cases.

Case 1. *$\mathcal{S}_E$ was built on E. Then, we have that $k = p_E$. In addition, by Beyer et al. [17] it holds that $\boldsymbol{E}[\widehat{d}_{AKMV}] = |E|_d$.*

Case 2. *$E = E_1 \circ E_2$ and $\mathcal{S}_E = \mathcal{S}_{E_1} \circ \mathcal{S}_{E_2}$. In this case, some hashes $h \in \mathcal{H}_E$ might have multiplicity $\eta_E(h) = 0$, e.g., when $\circ = \cap$. Then, $p_E \leq k$ reflects the reduced number of hashes we track from E. As in the previous case, $\boldsymbol{E}[\widehat{d}_{AKMV}] = |E|_d$ by [17].*

□

Note the following relationship between the distinct ratio of an AKMV and the number of tracked hashes k. Let $\mathcal{S}_{R.C}$ be an x-AKMV, i.e., has distinct ratio x, for attribute C of base table R. Then, $k = |\mathcal{H}_{R.C}| = x \cdot |R.C|_d$.

Next, we define a multiplication operation for AKMV sketches. Then, we show that the result of multiplying two AKMVs $\mathcal{S}_{R.\mathcal{C}}, \mathcal{S}_{T.D}$ gives an AKMV for the expression $\pi_C(R \bowtie_{C=D} T)$, or, equivalently, $\pi_D(R \bowtie_{C=D} T)$. Recall the illustration of our approach in Figure 8.1. As we will see, AKMV multiplication is the operation we need to combine AKMVs such that they approximate two-way join results.

Definition 8.2.1. *Let $\mathcal{S}_{R.\mathcal{C}}, \mathcal{S}_{T.D}$ be two AKMVs. Their product*

$$\mathcal{S}_{R.\mathcal{C}} \cdot \mathcal{S}_{T.D}$$

gives a new AKMV $\mathcal{S}_E$, where $\mathcal{H}_E$ contains the k smallest hashes in $\mathcal{H}_{R.\mathcal{C}} \cup \mathcal{H}_{T.D}$ and the multiplicity of each $h \in \mathcal{H}_E$ is $\eta_E(h) = \eta_{R.\mathcal{C}}(h) \cdot \eta_{T.D}(h)$.

Lemma 8.2.2. *Let $\mathcal{S}_E = \mathcal{S}_{R.\mathcal{C}} \cdot \mathcal{S}_{T.D}$. Then, $\mathcal{S}_E$ has, in expectation, the properties of an AKMV for $\pi_C(R \bowtie_{C=D} T)$.*

Proof. Recall correlated sampling [86], which we described in Section 4.1. We first show that the AKMVs $\mathcal{S}_{R.\mathcal{C}}, \mathcal{S}_{T.D}$ are equivalent to compressed correlated samples, a term we define below. Then we show that $\mathcal{S}_E = \mathcal{S}_{R.\mathcal{C}} \cdot \mathcal{S}_{T.D}$ behaves, in expectation, like a compressed correlated sample.

A correlated sample $R' \subseteq R$ for attribute C of relation R is defined as the multiset

$$R' = \{r \in R : H(r.C) < s_R\}_b,$$

where $s_R \in [0, 1]$ is some expected sample rate.

Suppose we're only interested in the number of times each $r.C, r \in R'$ occurs in the correlated sample R'. Further suppose, instead of knowing $r.C$, we're satisfied with its hash $H(r.C)$. Then we can define the *compressed correlated sample* R'_{comp} as the set of pairs

$$R'_{\text{comp}} := \{\ (H(r.C),\ |\{r_i \in R' : r_i.C = r.C\}_b|) : r \in R'\},$$

where the set notation eliminates duplicate pairs. Observe that R'_{comp} matches the specification of AKMV $\mathcal{S}_{R.C}$. The only difference is that R'_{compHash} is defined in terms of the expected sample ratio s_R while $\mathcal{S}_{R.C}$ is defined in terms of its fixed size k. Clearly, the above reasoning also holds true for a correlated sample T' for attribute D of relation T. Hence, AKMVs on base relations are equivalent to compressed correlated samples.

It remains to show how $\mathcal{S}_E$ relates to compressed correlated samples. Denote by $concat(r, t)$ a function that concatenates two input tuples r, t. Then, by [86],

$$E' = R' \bowtie_{C=D} T' = \{concat(r, t) : r \in R', t \in T', r.C = t.D\}_b$$

is a correlated sample of $E = R \bowtie_{C=D} T$. Clearly, the compressed correlated sample of E' is

$$E'_{\text{comp}} = \{\ (H(e.C),\ |\{e_i \in E' : e_i.C = e.C\}_b|) : e \in E'\}$$

which does *not* match the hashes $\mathcal{H}_E$ and their multiplicities of $\mathcal{S}_E$ since (1) only $\mathcal{H}_E$ might contain hashes whose multiplicity is zero and (2) $\mathcal{H}_E$ might not contain all hashes contained in E'_{comp} for its fixed size k. However, by [18], $\mathcal{H}_E$ constitutes a uniform random sample of the hashes in E'_{comp}. Thus, the expected properties of $\mathcal{S}_E$ match those of the compressed correlated sample E'_{comp}, which matches the specification of an AKMV built on E. □

Hence, AKMV multiplication allows us to derive AKMVs for join results from base table AKMVs. However, note that the derived AKMV $\mathcal{S}_E$ is not necessarily equal to an AKMV constructed on $\pi_C(R \bowtie_{C=D} T)$, since only $\mathcal{S}_E$ might contain hashes with multiplicity zero.

Next, we observe that, in addition to NODV estimation, an AKMV $\mathcal{S}_E$ can be used to estimate the cardinality $c := |E|$, i.e., the number of not necessarily distinct values in E.

Lemma 8.2.3 (cardinality estimate)**.** *Let $M_E := \sum_{h \in \mathcal{H}_E} \eta_E(h)$ be the summed multiplicities of the hashes in $\mathcal{H}_E$ of AKMV $\mathcal{S}_E$. An estimate for the cardinality c of relational algebra expression E is*

$$\widehat{c}_{AKMV} := \frac{M_E}{k} \cdot \frac{k-1}{\max(\mathcal{H}_E)}$$

Proof. Note that for each $h \in \mathcal{H}_E$, the multiplicity $\eta_E(h)$ is the number of times a value $v \in E$ with $H(v) = h$ occurs in E, and this value is known by $\mathcal{S}_E$. Recall that by [18] $\mathcal{H}_E$ is a random sample of the hashed values of E. It follows that $\mathbf{E}[\frac{M_E}{p_E}] = \frac{|E|}{|E|_d}$, where $p_E := |\{h_i \in \mathcal{H}_E \mid \eta(h_i) > 0\}|$, is an unbiased estimate for the average multiplicity of the values in E. It then follows that

$$\mathbf{E}\left[\frac{M_E}{p_E} \cdot \widehat{d}_{AKMV}\right] = |E| = c.$$

Then, plugging in the definition of $\widehat{d}_{AKMV}$ shows that

$$\frac{M_E}{p_E} \cdot \widehat{d}_{AKMV} = \frac{M_E}{p_E} \cdot \frac{p_E}{k} \cdot \frac{k-1}{\max(\mathcal{H}_E)} = \frac{M_E}{k} \cdot \frac{k-1}{\max(\mathcal{H}_E)}.$$

□

Corollary 8.2.1. *$\widehat{c}_{AKMV}$ of $\mathcal{S}_{R.\mathcal{C}} \cdot \mathcal{S}_{T.D}$ is an unbiased estimate for*

$$|\pi_C(R \bowtie_{C=D} T)|$$

.

Proof. Denote by $\mathcal{S}_{E*}$ an AKMV built on $E* = \pi_C(R \bowtie_{C=D} T)$. By Lemma 8.2.3, it holds that $\mathbf{E}[\widehat{c}_{AKMV}(\mathcal{S}_{E*})] = |E*|$.

Further, let $\mathcal{S}_E = \mathcal{S}_{R.\mathcal{C}} \cdot \mathcal{S}_{T.D}$. By Lemma 8.2.2, it holds that $\mathbf{E}[\widehat{c}_{AKMV}(\mathcal{S}_E)] = \mathbf{E}[\widehat{c}_{AKMV}(\mathcal{S}_{E*})]$.

Thus, we have that $\mathbf{E}[\widehat{c}_{AKMV}(\mathcal{S}_E)] = |E*|$. □

Due to Corollary 8.2.1 the size of an equi-join can be estimated from two AKMVs constructed for the join attributes in the base tables. Note that $|\pi_C(R \bowtie_{C=D} T)| = |\pi_D(R \bowtie_{C=D} T)| = |R \bowtie_{C=D} T|$.

Example: Suppose $\mathcal{S}_{R.\mathcal{C}} = (\{0.0002 < 0.003 < 0.015 < 0.05\}, (3, 5, 4, 8))$ and $\mathcal{S}_{T.D} = (\{0.0002 < 0.003 < 0.008 < 0.015\}, (10, 10, 10, 10))$. The product of $\mathcal{S}_{R.\mathcal{C}} \cdot \mathcal{S}_{T.D}$ is $\mathcal{S}_E = (\{0.0002 < 0.003 < 0.008 < 0.015\}, (30, 50, 0, 40))$. Since $M_E = 120$ is the sum of the multiplicities, the join size $|R \bowtie_{C=D} T|$ is estimated as $\widehat{c}_{AKMV} = 120/4 \cdot (4-1)/0.015 = 6000$.

8.3 Translation Grids

This section presents the *translation grid*, a data structure that approximates the joint existence distribution (JED), which we first mentioned in Section 8.1. In the following, we give a formal definition of JED. Suppose we have a relation R with attributes/attribute sets A and B. Then, the JED of $(R.A, R.B)$ is defined as the characteristic function

$$\mathbf{1}_{\pi_{A,B}(R)}(a, b) = \begin{cases} 1, & (a, b) \in \pi_{A,B}(R) \\ 0, & \text{else,} \end{cases}$$

Algorithm 4 Translation grid construction.

CONSTRUCT($R, A, B,\ m,\ n,\ hA_{max}, hB_{max}$)
Input: Relation R,
attribute sets A, B of R
two numbers $m,\ n$
largest hash values hA_{max}, hB_{max} in AKMVs of $R.A$ and $R.B$
Output: Translation grid $\mathcal{T}(R.A, R.B)$
1 Let $\mathcal{T}$ be a translation grid of $m \times n$ tiles
2 **for** $t \in R$
3 $hA = H(t.A)$
4 $hB = H(t.B)$
5 **if** $hA > hA_{max}$ **or** $hB > hB_{max}$: **continue**
6 $tile = \mathcal{T}$.TILE_REF(hA, hB)
7 $tile.\mathcal{B}_A$.INSERT(hA)
8 $tile.\mathcal{B}_B$.INSERT(hB)
9 **return** $\mathcal{T}$

where a and b are from the domains of attributes A and B, respectively. That is, $\mathbf{1}_{\pi_{A,B}(R)}$ indicates whether a combination of A, B values occurs in R. To approximate the JED of $(R.A, R.B)$, different approaches could be thought of. We present translation grids as our approach to approximate JEDs. Translation grids entirely operate on hashed attribute values of $R.A$ and $R.B$. In particular, those hashes that exist in AKMVs $\mathcal{S}_A, \mathcal{S}_B$ of $R.A$, $R.B$, respectively. Then, translation grids connect those hashes from $\mathcal{S}_A, \mathcal{S}_B$ that correspond to combinations of $R.A, R.B$ values that occur in R.

The example query from the introduction describes a situation where translation grids are useful. Given AKMVs for each column $D1.A, D2.B, F.A, F.B$, we can apply AKMV multiplication to obtain AKMVs $\mathcal{S}_{D1.A \bowtie F.A}$, $\mathcal{S}_{D2.B \bowtie F.B}$ for the two-way joins. Then, the translation grid connects those hashes in $\mathcal{S}_{D1.A \bowtie F.A}$ with hashes in $\mathcal{S}_{D2.B \bowtie F.B}$ that exist in the JED of $(F.A, F.B)$.

8.3.1 Structure

A translation grid $\mathcal{T}(R.A, R.B)$ is a two-dimensional grid of $m \times n$ *tiles*. For each tuple $t \in R$, the pair of hashes $(H(t.A), H(t.B))$, where H is the same hash function as for AKMV, maps to one tile. Each tile maintains two Bloom filters $\mathcal{B}_{R.A}, \mathcal{B}_{R.B}$. All $(H(t.A), H(t.B))$ corresponding to the same tile are inserted into the respective Bloom filters $\mathcal{B}_{R.A}, \mathcal{B}_{R.B}$ of that tile. As we will see, the fact that we insert hashes into the Bloom filters of the tiles of translation grid $\mathcal{T}(R.A, R.B)$, allows us to test whether a hash pair (h_A, h_B) from two AKMVs $\mathcal{S}_{R.A}, \mathcal{S}_{R.B}$ corresponds to attribute values in R.

8.3.2 Construction

The construction of translation grid $\mathcal{T}(R.A, R.B)$ happens in a single run over relation R. The construction of $\mathcal{T}(R.A, R.B)$ takes place *after* the con-

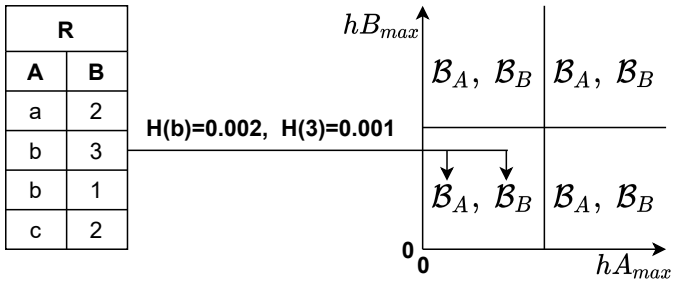


Figure 8.2: Construction of translation grid.

struction of AKMVs $\mathcal{S}_{R.A}, \mathcal{S}_{R.B}$. Algorithm 4 describes the translation grid construction process. For an illustration, see Figure 8.2. The arguments are relation R, with attribute sets A and B, and the desired dimensions of $\mathcal{T}(R.A, R.B)$ specified by m and n. In addition, the parameters hA_{max} and hB_{max} are the largest hash values found in any AKMV of $S.A$ and $S.B$, respectively. In line 1, $\mathcal{T}(R.A, R.B)$ is initialized with the bits of the two Bloom filters of each tile set to zero. Then, we iterate over each tuple $t \in R$ and hash each $t.A$ and $t.B$. In line 5, all hash pairs (hA, hB) with $hA{>}hA_{max} \vee hB{>}hB_{max}$ are discarded, i.e., we discard all hash combinations that are definitely not tracked by AKMVs. Filtering unnecessary (hA, hB) values helps to significantly reduce the size of a translation grid, or, equivalently, increase the accuracy it delivers for a given space consumption. For the remaining hash pairs (hA, hB), the corresponding tile is identified, and each hash is inserted in its respective Bloom filter. Finally, $\mathcal{T}(R.A, R.B)$ is returned.

Example: Suppose we build a translation grid $\mathcal{T}(R.A, R.B)$ with 2×2 tiles on the attribute values of R depicted in Figure 8.2. Suppose $hA_{max} = 0.05$ and $hB_{max} = 0.3$. Let $hA_{border} = hA_{max}/2 = 0.025$ define the boundary of the tiles in the $R.A$ dimension, and $hB_{border} = hB_{max}/2 = 0.15$ in the $R.B$ dimension. With respect to the values of $R.A$, let $H('a') \mapsto 0.03$, $H('b') \mapsto 0.002$, $H('c') \mapsto 0.07$. And with respect to the values of $R.B$, let $H(1) \mapsto 0.2$, $H(2) \mapsto 0.45$, $H(3) \mapsto 0.001$. Then, the tuples $<' a', 2>$ and $<' c', 2>$ are discarded for the check in line 5 of Algorithm 4. As illustrated in Figure 8.2, the tuple $<' b', 3>$ is inserted into the bottom left tile. The tuple $<' b', 1>$ is inserted into the bottom right tile.

8.3.3 Translation

By *translating*, we denote the process of finding combinations of hashes from two input AKMVs $\mathcal{S}_{R.A}, \mathcal{S}_{R.B}$ that $\mathcal{T}(R.A, R.B)$ has seen during construction. All qualifying combinations of hashes are output in two new AKMVs $\mathcal{S}'_{R.A}, \mathcal{S}'_{R.B}$. Note that, in the context of multi-way join size estimation, the input AKMVs $\mathcal{S}_{R.A}, \mathcal{S}_{R.B}$ are each the result of an intersect or multiply operation with another AKMV.

Algorithm 5 Translation grid, translation of AKMVs.

```
TRANSLATE(T, S_{R.A}, S_{R.B})
    Input:  Translation grid T,
            two input AKMVs S_{R.A}, S_{R.B}
    Output: Two output AKMVs S'_{R.A}, S'_{R.B}
1   Let S'_{R.A}, S'_{R.B} be two empty AKMVs
2   for hA ∈ S_{R.A} with η_{R.A}(hA) > 0
3       for hB ∈ S_{R.B} with η_{R.B}(hB) > 0
4           tile = T.TILE_REF(hA, hB)
5           if tile.B_A.CONTAINS(hA) and tile.B_B.CONTAINS(hB)
6               S'_{R.A}.INSERT(hA, η_{R.A}(hA))
7               S'_{R.B}.INSERT(hB, η_{R.B}(hB))
8   return S'_{R.A}, S'_{R.B}
```

Algorithm 5 describes how to translate two input AKMVs using a translation grid. For each combination of hashes (hA, hB) with multiplicity greater zero, i.e., $\eta_{R.A}(hA) > 0 \wedge \eta_{R.B}(hB) > 0$, the corresponding tile is identified in line 4. Then, in line 5, we test if hA exists in $\mathcal{B}_A$ and if hB exists in $\mathcal{B}_B$ of the identified tile. Note that, as always with Bloom filters, false positives are possible. All combinations of hashes (hA, hB) that pass the test are tracked in the two output AKMVs $\mathcal{S}'_{R.A}, \mathcal{S}'_{R.B}$. Essentially, the output AKMVs $\mathcal{S}'_{R.A}, \mathcal{S}'_{R.B}$, as returned in line 8, are filtered versions of $\mathcal{S}_{R.A}, \mathcal{S}_{R.B}$. As we will see in the next section, $\mathcal{S}'_{R.A}, \mathcal{S}'_{R.B}$ are proper AKMVs that can be used for cardinality estimation.

Example: Let $\mathcal{T}(S.J1, S.J2)$ be a translation grid, defined as follows:

$\mathcal{B}_{J1}$:{.05}, $\mathcal{B}_{J2}$:{.7}	$\mathcal{B}_{J1}$:{.11}, $\mathcal{B}_{J2}$:{.21, .27}	$\mathcal{B}_{J1}$:{.22, .25}, $\mathcal{B}_{J2}$:{.21}
$\mathcal{B}_{J1}$:{.05}, $\mathcal{B}_{J2}$:{.1, }	$\mathcal{B}_{J1}$:{.11}, $\mathcal{B}_{J2}$:{.111}	$\mathcal{B}_{J1}$:{.25}, $\mathcal{B}_{J2}$:{.1}
$\mathcal{B}_{J1}$:{.05}, $\mathcal{B}_{J2}$:{.02}	$\mathcal{B}_{J1}$:{}, $\mathcal{B}_{J2}$:{}	$\mathcal{B}_{J1}$:{.22, .25}, $\mathcal{B}_{J2}$:{.007}

Further, let $\mathcal{S}_{J1} = \{< .05, 3 >, < .11, 5 >, < .22, 0 >, < .25, 1 >\}$ be an AKMV for $R \bowtie_{R.J1=S.J1} S$, derived from two AKMVs via multiplication in the sense of Definition 8.2.1. Accordingly, let $\mathcal{S}_{J2} = \{< .02, 2 >, < .1, 0 >, < .21, 2 >, < .27, 1 >, \}$ be AKMV for $S \bowtie_{S.J2=T.J2} T$. Then, TRANSLATE($\mathcal{T}(S.J1, S.J2)$, $\mathcal{S}_{J1}, \mathcal{S}_{J2}$) outputs the new AKMVs $\mathcal{S}'_{J1} = \{< .05, 6 >, < .11, 5 >, < .25, 2 >\}$ and $\mathcal{S}'_{J2} = \{< .02, 2 >, < .21, 3 >, < .27, 1 >\}$.

8.3.4 Estimation Properties of Translation Grids

This section presents two Lemmata that describe how translation grids can be used, in conjunction with AKMVs, for cardinality estimation.

First, we define an ideal translation grid to which we refer as a *lossless translation grid*.

Definition 8.3.1 (Lossless translation grid)**.** *A lossless translation grid $\mathcal{T}(R.A, R.B)$, correctly determines for each $h_A \in \mathcal{S}_{R.A}, h_B \in \mathcal{S}_{R.B}$ iff (h_A, h_B) corresponds to a tuple in R. There are no false-positives in a lossless translation grid.*

A lossy translation grid is one that is not lossless (produces false-positives).

In the following lemma, we combine a lossless translation grid with two AKMVs. To understand the Lemma, recall that by a x-AKMV we refer to an AKMV that tracks a fraction x of the hashes seen during construction.

Lemma 8.3.1. *Let $\mathcal{T}(R.A, R.B)$ be a lossless translation grid of relation R with $|R|=|R.A|_d=|R.B|_d$. As before, denote by $\widehat{d}_{AKMV}$ and $\widehat{c}_{AKMV}$ the NODV estimate and the cardinality estimate (cf. Lemma 8.2.3) obtained from an AKMV $\mathcal{S}$, respectively. Suppose a x-AKMV $\mathcal{S}_{R.A}$, $x \in [0,1]$, and a 1.0-AKMV $\mathcal{S}_{R.B}$ are translated by $\mathcal{T}(R.A, R.B)$. Then*

$$\mathbf{E}[\widehat{d}_{AKMV}(\mathcal{S'}_{R.A})] = \mathbf{E}[\widehat{c}_{AKMV}(\mathcal{S'}_{R.A})] = |R|,$$

holds for the output AKMV $\mathcal{S'}_{R.A}$.

Proof. Since $\mathcal{S}_{R.B}$ is a 1.0-AKMV, i.e., tracks all hashes seen during construction, it follows that each hash $h \in \mathcal{S}_{R.A}$ necessarily finds all its matches in $\mathcal{T}(R.A, R.B)$. In addition, since $\mathcal{T}(R.A, R.B)$ is a lossless translation grid, there are no false positive matches for $h \in \mathcal{S}_{R.A}$. Thus, $\mathcal{S'}_{R.A} = \mathcal{S}_{R.A}$.

Finally, $\mathbf{E}[\widehat{c}_{AKMV}(\mathcal{S}_{R.A})] = |R|$ holds by Lemma 8.2.3, and $\mathbf{E}[\widehat{c}_{AKMV}(\mathcal{S}_{R.A})] = \mathbf{E}[\widehat{d}_{AKMV}(\mathcal{S}_{R.A})]$ since neither $R.A$ nor $R.B$ contain duplicates. □

Note that the above lemma does not hold for $\mathcal{S'}_{R.B}$. However, we clearly do not want to rely on 1.0-AKMVs. Luckily, the following lemma comes to the rescue.

Lemma 8.3.2. *Suppose the same setting as in Lemma 8.3.1, but this time $\mathcal{S}_{R.B}$ is a y-AKMV, where $y \in [0,1]$. It follows that*

$$\mathbf{E}\left[\frac{\widehat{c}_{AKMV}(\mathcal{S'}_{R.A})}{y}\right] = |R|$$

.

Proof. In expectation, each $h \in \mathcal{S}_{R.A}$ finds a ratio of y of its matches in $\mathcal{T}(R.A, R.B)$. Since $\mathcal{T}(R.A, R.B)$ is a lossless translation grid no false positives are found. Thus, we must scale up $\widehat{c}_{AKMV}(\mathcal{S'}_{R.A})$ by y. □

Note that, from symmetry, also $\mathbf{E}\left[\frac{\widehat{c}_{AKMV}(\mathcal{S'}_{R.B})}{x}\right] = |R|$ holds. In our evaluation, we present an experiment to illustrate Lemma 8.3.2.

Observation: Lemma 8.3.2 is applicable for multi-way join size estimation. For instance, let $E1, E2$ be relational expressions with key attributes A, B, respective, and $\mathcal{T}(F.A, F.B)$ a translation grid for a relation F with foreign key attributes A, B. Assume AKMVs $\mathcal{S}_{E1.A}, \mathcal{S}_{F.A}, \mathcal{S}_{F.B}, \mathcal{S}_{E2.B}$ exist. Applying Lemma 8.3.2 to the output AKMVs of TRANSLATE($\mathcal{T}(F.A, F.B)$, $\mathcal{S}_{F.A} \cdot \mathcal{S}_{E1.A}$, $\mathcal{S}_{F.B} \cdot \mathcal{S}_{E2.B}$) gives an estimate for $|E1 \bowtie_{E1.A=F.A} F \bowtie_{F.B=E2.B} E2|$. Section 8.4 presents an experiment on multi-way join size estimation.

Corollary 8.3.1. *An indicator for a lossy translation grid is to test if*

$$\frac{\widehat{c}_{AKMV}(\mathcal{S}'_{R.A})}{y} \not\approx \frac{\widehat{c}_{AKMV}(\mathcal{S}'_{R.B})}{x}$$

.

Proof. From symmetry of Lemma 8.3.2, it follows that $\mathbf{E}\left[\frac{\widehat{c}_{AKMV}(\mathcal{S}'_{R.A})}{y}\right] = \mathbf{E}\left[\frac{\widehat{c}_{AKMV}(\mathcal{S}'_{R.B})}{x}\right]$. Corollary 8.3.1 is essentially the contraposition. The contraposition holds true if one the premises of Lemma 8.3.2 are violated, one of which is that $\mathcal{T}(R.A, R.B)$ is a lossless translation grid. □

Note that Corollary 8.3.1 makes a statement under uncertainty. For instance, $\mathcal{T}(R.A, R.B)$ may be a lossless translation grid but $\frac{\widehat{c}_{AKMV}(\mathcal{S}'_{R.A})}{y}$ differs significantly from $\frac{\widehat{c}_{AKMV}(\mathcal{S}'_{R.B})}{x}$ simply because, by chance, one estimate deviates significantly from its expected value.

8.4 Evaluation

In our evaluation, we present multiple experiments. Our experiments are designed to demonstrate the validity of the statements from sections 8.2 and 8.3 and to analyze the accuracy of our approach for result size estimation of queries involving multiple joins.

Experiment 1:
Let R: [{A, B}] with $|R|$=$|R.A|_d$=$|R.B|_d$=1000. We construct $\mathcal{S}_{R.A}$ as 0.3-AKMV and $\mathcal{S}_{R.B}$ is a y-AKMV, where y takes the values specified in Table 8.1. Recall the relationship between the distinct ratio and number of tracked hashes described below Lemma 8.2.1. Hence, $\mathcal{S}_{R.A}$ tracks 300 hashes and $\mathcal{S}_{R.B}$ tracks $y \cdot 1000$ hashes, i.e., between 100 and 1000 hashes. The lossless translation grid $\mathcal{T}(R.A, R.B)$ is approximated by $|R.A|_d \times |R.B|_d$ tiles. We denote

$$cA := \widehat{c}_{AKMV}(\mathcal{S}'_{R.A})$$

and

$$cB := \widehat{c}_{AKMV}(\mathcal{S}'_{R.B}).$$

Clearly, if Lemma 8.3.2 holds, we expect

$$cA/y \approx cB/0.3 \approx 1000.$$

Table 8.1 shows the result. Indeed, even for y=0.1, we have $cA/y \approx 1000$. For comparison, we include cA, cB. Note how cA decreases as y decreases, since each $h \in \mathcal{S}'_{R.A}$ finds fewer and fewer matches in $\mathcal{T}(R.A, R.B)$.

How many hash pairs (hA, hB) are inserted into $\mathcal{T}(R.A, R.B)$ during construction? By line 5 in Algorithm 4, only hashes possibly tracked by AKMVs are inserted into $\mathcal{T}(R.A, R.B)$. Hence, we expect $0.3 \cdot y \cdot |R|$ many inserts, cf. the

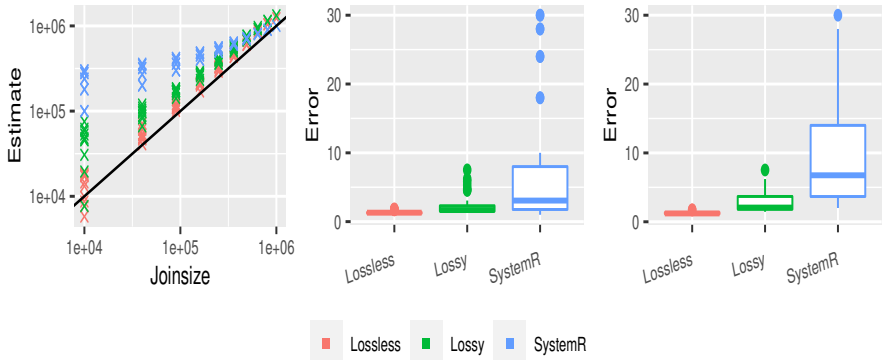


Figure 8.3: Results of Experiment 2: Scatterplot (left), errors over all queries (middle), errors over selective queries (right).

last column of Table 8.1. The second to last column shows the actual number of inserts. Observe that the actual number is both small and close to the expected number. Note that the number of distinct inserts is the same since neither $R.A$ nor $R.B$ contains duplicates.

Experiment 2:
Recall the star query from the introduction. For varying selection predicates, we estimate the query result cardinality, i.e.,

$$|\sigma_{p_{D1}}(D1) \bowtie_{D1.A=F.A} F \bowtie_{F.B=D2.B} \sigma_{p_{D2}}(D2)|.$$

The dimension tables have $|D1|$=1000 and $|D2|$=2000 rows, respectively, and the attributes A, B refer to their respective primary key. The fact table

$$F := \{[i,j] | i \in [0,|D1|), j \in [0,|D2|), \frac{i}{|D1|} \geq \frac{j}{|D2|}\}$$

has the skewed foreign key attributes $F.A, F.B$ that are correlated with each other. Its size is $|F|$=1,000,000. The cardinality of a query result is influenced

y	cA	cA/y	cB	$cB/0.3$	ins.	exp. ins.
1	1075.48	1075.48	300.471	1001.57	300	300
0.9	949.591	1055.1	291.39	971.3	265	270
0.8	838.086	1047.61	295.724	985.746	234	240
0.7	715.79	1022.56	288.319	961.063	200	210
0.6	600.688	1001.15	287.529	958.431	168	180
0.5	504.093	1008.19	288.892	962.974	141	150
0.4	414.077	1035.19	297.449	991.497	116	120
0.3	316.527	1055.09	296.691	988.969	87	90
0.2	209.791	1048.95	317.068	1056.89	58	60
0.1	111.541	1115.41	326.231	1087.44	30	30

Table 8.1: Results of Exper. 1. As expected $\frac{cA}{y} \approx \frac{cB}{0.3} \approx 1000$.

by the selection predicates

$$p_{D1} \equiv D1.A{<}c_1,\ p_{D2} \equiv D2.B{\geq}c_2,$$

where we compute results for all combinations with non-zero cardinality of $c_1{\in}\{100, 200, \ldots, 1000\}$, $c_2{\in}\{0, 200, 400, \ldots, 1800\}$. Hence, the true cardinality varies between 10k and 1M. We make the following assumption: Since $D1, D2$ are small, AKMVs $\mathcal{S}_{\pi_r(\sigma_{p_{D1}}(D1))}, \mathcal{S}_{\pi_t(\sigma_{p_{D2}}(D2))}$ can be built on the fly. Since F is large, $\mathcal{S}_{F.A}, \mathcal{S}_{F.B}$ and $\mathcal{T}(F.A, F.B)$ must be computed offline, i.e. not per query. All AKMVs are of size k=100. An estimate for the query result size is obtained in three steps:

1. Compute $\mathcal{S}_{J1} := \mathcal{S}_{F.A} \cdot \mathcal{S}_{\pi_r(\sigma_{p_{D1}}(D1))}$ and $\mathcal{S}_{J2} := \mathcal{S}_{F.B} \cdot \mathcal{S}_{\pi_t(\sigma_{p_{D2}}(D2))}$.
2. Obtain $(\mathcal{S'}_{J1}, \mathcal{S'}_{J2}) = \textsc{Translate}(\mathcal{T}(F.A, F.B), \mathcal{S}_{J1}, \mathcal{S}_{J2})$.
3. Estimate the cardinality as $\widehat{c}_{AKMV}(\mathcal{S'}_{J1})/\widehat{r}$, where $\widehat{r}$ is the estimated distinct ratio of $\mathcal{S'}_{J2}$, cf. Lemma 8.2.1.

In Figure 8.3, for each query, we show three different cardinality estimates. (1) An estimate obtained by a lossless translation grid, which tracks only $5200 \approx 0.5\% \cdot |F|$ pairs of hashes. The size of only $5200{\cdot}2{\cdot}4B = 41.6KB$ demonstrates the practicability of lossless translation grids. (2) A lossy 100×100 translation grid, in which only 2600 tiles are actually used. Each tile contains only two 32-bit Bloom filters. Hence, the used tiles consume only $2600{\cdot}2{\cdot}4B = 20.8KB$, corresponding to only 0.26% of F's memory footprint, assuming each tuple in F consists of 2 integers of 32-bit. (3) For comparison, we show the System R [79] estimator, as implemented in Postgres version 13.2. In Figure 8.3, the scatterplot on the left shows the correlation between the observed and estimated cardinality. The best estimates are obtained when using the lossless translation grid, the lossy translation grid produces slightly worse estimates. System R significantly overestimates, since its assumptions do not hold. The boxplots in the middle of Figure 8.3 illustrate the q-error for each estimator from the scatterplot. The boxplots on the right are restricted to only the challenging queries with selectivity of p_{D1} less than 0.5. Note how the errors significantly worsen for System R when looking only at the more selective queries. The estimates by the lossless and lossy translation grid are less affected.

Experiment 3:
This experiment tests Corollary 8.3.1. We reuse the setup and notation from Experiment 1. For translation grids with different dimensions dimA $\times$ dimB, we use x-AKMV $\mathcal{S}_{R.A}$ and y-AKMV $\mathcal{S}_{R.B}$ to test if

$$\frac{cA}{y} \not\approx \frac{cB}{x} \implies \textit{lossy}.$$

The results are shown in Table 8.2. Recall that a lossy translation grid produces false positives. Hence, when there are more inserts into the output AKMVs,

dimA	dimB	x	y	k_A	k_B	cA/y	cB/x	ins.	found	lossy
1000	250	0.30	0.80	300	800	1047.61	985.746	234	234	×
1000	62	0.30	0.80	300	800	1047.61	985.746	234	234	×
1000	31	0.30	0.80	300	800	1047.61	994.207	234	240	✓
1000	15	0.30	0.80	300	800	1047.61	998.438	234	249	✓
1000	7	0.30	0.80	300	800	1047.61	1070.36	234	303	✓
250	7	0.30	0.80	300	800	1047.61	1104.2	234	356	✓
7	7	0.30	0.80	300	800	1312.88	3380.31	234	58753	✓
125	125	0.30	0.10	300	100	1115.41	1087.44	30	30	×
7	7	0.30	0.10	300	100	2615.44	1499.91	30	210	✓
125	125	0.10	0.10	100	100	1044.93	1133.66	11	11	×
7	7	0.10	0.10	100	100	1763.1	1587.12	11	36	✓
7	62	0.10	0.10	100	100	1763.1	1133.66	11	21	✓

Table 8.2: Results of Exper. 3. As expected $\frac{cA}{y} \not\approx \frac{cB}{x} \implies$ *lossy*.

$\lvert\pi^D_A(R)\rvert$	$\lvert\pi^D_B(R)\rvert$	TGdim	x	$\hat{x}$	hashcount	$\hat{d}_{AKMV}(\mathcal{S}'_{R.A})$	k_A	k_B
150	133	2000	1.00	1.00	150	149.98	300	400
150	133	20	1.00	1.00	150	149.98	300	400
500	333	2000	0.20	0.19	55	282.42	100	100
500	333	20	0.20	0.19	55	282.42	100	100
500	333	20	0.10	0.08	13	153.97	50	50
500	333	20	0.04	0.05	4	75.08	20	30
333	250	20	0.15	0.17	27	163.24	50	50
333	250	20	**0.03**	**1.29**	**1**	**0.78**	10	10
250	200	20	0.20	0.20	38	190.80	50	50
34	100	5	0.29	0.29	10	35.07	10	10
33	41	5	0.30	0.29	10	34.65	10	10
31	33	5	0.97	0.99	30	30.31	30	30
31	33	20	0.97	0.99	30	30.31	30	30
31	33	100	0.97	0.99	30	30.31	30	30

Table 8.3: Quality of distinct ratio estimator.

denoted by *found*, than values stored in the translation grid, denoted by *ins.*, we are sure the translation grid is lossy. The last column Table 8.2 marks the lossy translation grids. As expected, when $\frac{cA}{y} \not\approx \frac{cB}{x}$ then $lossy = \checkmark$. Also, observe the rate at which *ins.* decreases in x and y. Finally, note that from x and y it follows that $\mathcal{S}_{R.A}$ and $\mathcal{S}_{R.B}$ track $k_A = x \cdot |R.A|_d$ and $k_B = y \cdot |R.B|_d$ hashes, respectively. For convenience, the table also shows k_A and k_B.

Experiment 4:
Recall that our estimators use the distinct ratio x of an x-AKMV. As we have seen in Lemma 8.2.1, for AKMVs that were not built on base relations but are, e.g., the product AKMV multiplication, the distinct ratio can be estimated as $\mathcal{S}_E$ is $\frac{p_E}{\hat{d}_{AKMV}}$. This experiment evaluates the accuracy of this estimate.

Let $R : [\{A, B\}]$ with $|R| = 1000$. Table 8.3 shows, for relations with different attribute domain sizes $|\pi^D_A(R)|$, $|\pi^D_B(R)|$ for which a translation grid of

dimension $TGdim \times TGdim$ is built, the distinct ratio x and its estimate $\hat{x}$ of the output AKMV ${\mathcal{S}'}_{R.A}$. *hashcount* denotes the number of hashes stored in the output AKMV ${\mathcal{S}'}_{R.A}$. The last two columns k_A and k_B show the number of hashes $\mathcal{S}_{R.A}$ and $\mathcal{S}_{R.B}$ track, respectively. Note that the estimates have a good accuracy, except for the case highlighted in bold. Clearly, $\hat{x}$ is an overestimate since the distinct ratio must be in $[0, 1]$. However, it comes by no surprise that this estimate is inaccurate, since it is derived from an AKMV with only one hash value stored.

Experiment 5:
We present an experiment where we estimate the result cardinality of the following chain query with 4 relations and one selection predicate:

$$\sigma_{p_1}(r_1) \bowtie_{r_1.k=r_2.fk} r_2 \bowtie_{r_2.k=r_3.fk} r_3 \bowtie_{r_3.k=r_4.fk} r_4,$$

where $|r_1| = 1000, |r_2| = 3000, |r_3| = 9000, |r_4| = 27000$ and $p_1 := r1.k < \frac{|r1|}{4}$. The key $r_i.k$ of each relation r_i contains the values $[0, |r_i| - 1]$ and all foreign keys fk are uniformly distributed.

In this experiment, we use two lossless translation grids $\mathcal{T}(r_2.k, r_2.fk)$ and $\mathcal{T}(r_3.k, r_3.fk)$. In addition, we have base table AKMVs with $k = 300$. The goal is to obtain an estimate $\widehat{|r1234|}$ for the result cardinality of the above chain query, to which we refer as $|r1234|$. Recall that we first use AKMV multiplication to combine base table AKMVs. We denote $\mathcal{S}_{rij} = \mathcal{S}_{ri} \cdot \mathcal{S}_{rj}$. Further recall that TRANSLATE takes two AKMVs $\mathcal{S}_{rij}, \mathcal{S}_{rjk}$ as input and outputs two modified AKMVs $\mathcal{S}'_{rij}, \mathcal{S}'_{rjk}$. Conceptually (the actual parameterization of TRANSLATE differs from the following), there are two orderings of TRANSLATE to obtain $\widehat{|r1234|}$. We either compute

$$O1 := \textsc{Translate}(\textsc{Translate}(\mathcal{S}_{r12}, \mathcal{S}_{r23}), \mathcal{S}_{r34})$$

or

$$O2 := \textsc{Translate}(\mathcal{S}_{r12}, \textsc{Translate}(\mathcal{S}_{r23}, \mathcal{S}_{r34})).$$

Since each TRANSLATE operation gives two output AKMVs, there are 4 estimates per TRANSLATE ordering to obtain $\widehat{|r1234|}$.

In the following, we use T as a short hand notation for the TRANSLATE function and denote by $\mathrm{T}(\mathcal{S}_{rij}, \mathcal{S}_{rjk}).{\mathcal{S}'}_{rij}$ access to the output AKMV $\mathcal{S}'_{rij}$. Further denote by $|\mathrm{T}(\mathcal{S}_{rij}, \mathcal{S}_{rjk}).{\mathcal{S}'}_{rij}|$ the join size estimate obtained from ${\mathcal{S}'}_{rij}$. In the following we report the different estimates. The true join size is $|r1234| = 6750$. The 4 estimates for ordering $O1$ are:

- $|\mathrm{T}(\mathrm{T}(\mathcal{S}_{r12}, \mathcal{S}_{r23}).{\mathcal{S}'}_{r12}, \mathcal{S}_{r34}).{\mathcal{S}''}_{r12}| = 21564$
- $|\mathrm{T}(\mathrm{T}(\mathcal{S}_{r12}, \mathcal{S}_{r23}).{\mathcal{S}'}_{r23}, \mathcal{S}_{r34}).{\mathcal{S}''}_{r23}| = 2666$
- $|\mathrm{T}(\mathrm{T}(\mathcal{S}_{r12}, \mathcal{S}_{r23}).{\mathcal{S}'}_{r12}, \mathcal{S}_{r34}).{\mathcal{S}'}_{r34}| = 32346$
- $|\mathrm{T}(\mathrm{T}(\mathcal{S}_{r12}, \mathcal{S}_{r23}).{\mathcal{S}'}_{r23}, \mathcal{S}_{r34}).{\mathcal{S}'}_{r34}| = 10449$

While the 4 estimates for ordering $O2$ are:

- $|\mathrm{T}(\mathcal{S}_{r12}, \mathrm{T}(\mathcal{S}_{r23}, \mathcal{S}_{r34}).\mathcal{S}'_{r23}).\mathcal{S}'_{r12}| = 1650$
- $|\mathrm{T}(\mathcal{S}_{r12}, \mathrm{T}(\mathcal{S}_{r23}, \mathcal{S}_{r34}).\mathcal{S}'_{r34}).\mathcal{S}'_{r12}| = 3680$
- $|\mathrm{T}(\mathcal{S}_{r12}, \mathrm{T}(\mathcal{S}_{r23}, \mathcal{S}_{r34}).\mathcal{S}'_{r23}).\mathcal{S}''_{r23}| = 4651$
- $|\mathrm{T}(\mathcal{S}_{r12}, \mathrm{T}(\mathcal{S}_{r23}, \mathcal{S}_{r34}).\mathcal{S}'_{r23}).\mathcal{S}''_{r34}| = 3680$

As in the previous experiments, we expected all estimates to be approximately equal. However, observe that the estimates have significantly diverged. The smallest estimate is 1650, while the largest is 32346. Similar divergence occurs for different choices of p_1 and selection predicates on other tables. The two questions that follow are:

1. Why do the estimates diverge?
2. Can we combine the different estimates into one?

Regarding question one, we note that each output AKMV is derived differently. Hence, the diverged estimates can be explained by the tracked hashes, and corresponding multiplicities, that each output AKMV has left after two rounds of TRANSLATE. A similar problem occurs in join size estimation with AGMS sketches [75]. Each AGMS is essentially an array of counters. Each counter can be regarded as an estimator and all counters have the same expected value, yet the observed counter values usually differ significantly. Clearly, this is due to the variance of the estimator. As Huang et al. [39] have shown, join size estimation always comes with a high variance. Let $J = T_1 \bowtie_{T_1.X=T_2.Y} T_2$ be the join of tables T_1, T_2. Then, by Huang et al. [39], the best estimator for $|J|$ that is based on a synopses of b bits, e.g., a sample or a sketch, must have variance at least proportional to $\min(|T_1|, |T_2|) \cdot |T_1.X \cap T_2.Y|/b$.

Regarding question 2, we again look at AGMS sketches. In AGMS, the solution to combine the many different estimates is to first compute multiple averages, each computed over a chunk of counters. Then, the median over the averages serves as the final estimate. It remains open what the best way is to combine the different estimates in our case. A simple approach is to ignore the two largest and two smallest estimates and average over the remaining estimates obtained by the output estimates. We refer to this approach as the *incomplete average estimate* (IAE). For the above query, the IAE is $\widehat{|r1234|} = 5615$, with a q-error of around only 1.2. Motivated by this result, we repeat the above experiment for AKMV parameter $k \in \{500, 300, 100\}$ and different predicates $p_1 = r1.k < \frac{|r1|}{v}$, where $v \in \{2, 3, \ldots, 10\}$. To estimate the result cardinality $|r1234|$, we apply the IAE. We report the results in Table 8.4. Observe that for large AKMVs, $k = 500$, IAE yields good estimates in all cases, even for queries where p_1 is selective, which is the case when v is large. For AKMVs with $k = 300$, we observe good estimates, except for the selective queries with $v \in \{8, 9, 10\}$. For small AKMVs with $k = 100$, note that no estimate can be

k	v	$\lvert r1234\rvert$	$\widehat{\lvert r1234\rvert}$	q-error
500	2	5469	13500	2.46
500	3	3853	8991	2.33
500	4	3361	6750	2.00
500	5	2432	5400	2.22
500	6	1767	4482	2.53
500	7	1595	3834	2.40
500	8	1100	3375	3.06
500	9	749	2997	3.99
500	10	708	2700	3.80
300	2	9389	13500	1.43
300	3	8608	8991	1.04
300	4	5615	6750	1.20
300	5	3732	5400	1.44
300	6	3377	4482	1.32
300	7	855	3834	4.48
300	8	5	3375	596.82
300	9	5	2997	531.30
300	10	5	2700	478.65
100	2	107256	13500	7.94
100	3	107771	8991	11.98
100	4	116979	6750	17.33
100	5	113203	5400	20.96
100	6	95703	4482	21.35
100	7	40	3834	95.21
100	8	-	3375	-
100	9	-	2997	-
100	10	-	2700	-

Table 8.4: Average over output AKMV estimates. Two largest/smallest estimates are ignored.

obtained for the selective queries with $v \in \{8, 9, 10\}$, since no tracked hashes are left in the output AKMVs. We conclude that, in cases where the different estimates of the output AKMVs have diverged, the IAE seems to be a reasonable heuristic.

8.5 Summary

We presented a novel approach to estimate query result sizes for queries containing multiple joins. Our approach relies on two novelties. (1) New operations for AKMV sketches, in particular distinct ratio estimation, AKMV multiplication for two-way join approximation, and cardinality estimation. (2) A novel data structure called *translation grid* that partitions the hash space of AKMVs and tracks combinations of hashes from two AKMVs in such a way that they approximate the joint existence distribution of two attributes/attribute sets. In our evaluation, we analyzed the accuracy of our approach and experimentally

tested the validity of certain properties of translation grids. We demonstrated that our approach allows to estimate the cardinality of three-way joins with good accuracy. In addition, we discussed a join query over four relations where we have seen that even if the estimates of different output AKMVs diverge, we can combine them into one estimate, even though it remains open what approach to combine diverged estimates is best.

Chapter 9

Query Featurization

9.1 Introduction

For some years now, Machine Learning (ML) has been applied to the cardinality estimation problem [38, 47, 93, 94]. However, surprisingly, there has been very little research yet on how to present queries to a machine learning model. Machine learning models do not simply consume SQL strings. Instead, a SQL string is transformed into a numerical representation. This transformation is called *query featurization* and is defined by a query featurization technique (QFT). This chapter is concerned with QFTs for queries with many selection predicates. In particular, we consider queries that contain both predicates over different attributes and multiple predicates per attribute.

First, we provide some background knowledge. In general, ML means *arbitrary function approximation*. The function that underlays the cardinality estimation problem in databases is

$$\textbf{query} \times \textbf{data} \rightarrow \textbf{cardinality} \tag{9.1}$$

Note that the data component often remains unmentioned even though it is relevant since the query result of `SELECT count(*) FROM R WHERE R.A < 5` differs, depending on the content in `R`. Nonetheless, in this chapter, we assume the data to be fixed and substitute Equation 9.1 for the two-step mapping

$$\textbf{query} \rightarrow \textbf{vector} \rightarrow \textbf{cardinality} \tag{9.2}$$

The mapping **query** $\rightarrow$ **vector** is denoted by *query featurization* and benefits from expert knowledge. Query featurization is necessary since machine learning models do not consume SQL strings. Instead, a numerical representation of the query is consumed. Techniques for query featurization are in the focus of this chapter. The second mapping **vector** $\rightarrow$ **cardinality** is the machine learning part and benefits from ML knowledge, in particular algorithm choice and parameter tuning.

Despite the fact that all learning-based cardinality estimation techniques need query featurization, most current approaches do not focus on the query featurization part [47, 93]. For instance, Kipf et al. learn the featurized representation of a query from simple per-predicate featurizations. However, learned query featurization leads to obfuscated query representations that ultimately result in sub-optimal query result size estimates. We argue that there is no need for learning-based query featurization and that a smart query featurization technique (QFT) can be identified and implemented. Hence, this chapter focuses on improved QFTs.

The contribution of this chapter is the result of a fruitful collaboration with PhD candidate Lucas Woltmann from TU Dresden. In particular, we present three novel QFTs. We compare their pros and cons, especially how well they represent queries and how different types of ML models benefit from good query featurization in terms of estimation accuracy and the number of queries needed for training. In particular, we present the following findings:

- A formal definition of *good* query featurization, to which we refer to as *lossless query featurization*. To the best of our knowledge, we are the first to define requirements for query featurization. The definition is presented in Section 9.3 and motivates the design of our QFTs.

- We present (one established and) two novel QFTs for conjunctive queries, i.e., queries whose predicates are connected by `AND`. Unlike [94], we consider queries with multiple predicates per attribute.

- We present one additional QFT for queries including both conjunctions as well as disjunctions, i.e., predicates connected by `OR`. To the best of our knowledge, we are the first to consider disjunctions in ML-based cardinality estimation.

- We provide an extensive evaluation showing the effect of QFT on the estimation accuracy of several established machine learning models. In particular, we consider MSCN [47], local neural networks [93], and gradient boosting [28, 59]. As part of our evaluation, we also consider the impact of QFTs on estimation accuracy under query drift and different numbers of training queries.

Outline. The structure of the chapter is as follows: In the next section, we outline preliminaries in terms of query featurization and machine learning. Then, in Section 9.3, we present our three new QFTs and discuss their advantages and limitations in detail. In Section 9.4, we focus on implementation details of our QFTs as part of learning-based cardinality estimators. Next, we evaluate our new QFTs under different ML models with two data sets in Section 9.5. In Section 9.6, we discuss how to generalize our approach, for instance to queries with string predicates and aggregations. Finally, we draw a conclusion and discuss future work.

9.2 Preliminaries

In this section, we first present techniques to obtain feature vectors from queries. In particular, we discuss existing query featurization techniques for selection predicates and join predicates. Then, we present how machine learning models use feature vectors, and briefly discuss two well-known machine learning models.

9.2.1 State of the Art in Query Featurization

In this section, we present existing techniques for query featurization techniques – QFTs. Recall that a QFT encodes a query $\mathcal{Q}$ into a numerical vector, named *feature vector*. This feature vector then serves as input to a machine learning model – in particular one used for cardinality estimation. In query featurization, the critical part is the encoding of the predicates and joins from input query $\mathcal{Q}$. This section first presents two approaches, Singular Predicate Encoding and Predicate Set Convolution, to featurize selection predicates. Then, we discuss how to featurize queries containing joins. A word on the jargon we use. We either say a QFT featurizes a query or we say a QFT encodes a query, or certain parts thereof, in a feature vector. We use the terms featurization and encoding interchangeably.

9.2.1.1 Selection Predicate Featurization

First, we focus on the encoding of selection predicates. We describe how existing approaches featurize simple predicates, i.e., a predicate that compares an attribute value to a literal using one of the comparison operators in $\{=, >, <, \geq, \leq, \neq\}$. In particular, we present the predicate featurization methods by [35, 47, 93]. All three methods featurize selection predicates in two steps. The first step is to featurize each individual predicate in a query. This step is the same in all three methods. Then, the featurizations of each predicate are combined. As we will see, in this second step, [35, 93] differ from [47].

To encode a predicate like `A > 5`, the predicate is split and encoded into three parts. (1) Attribute `A` is encoded in a one-hot vector, i.e., only one entry is set, like 001 for a relation with three attributes, (2) the literal `5` is encoded to $\frac{5-\min(A)}{\max(A)-\min(A)}$, which is always a number in $[0, 1]$, and (3) the comparison operator `>` is encoded to a binary vector with 3 entries, using the mapping outlined in the following table:

comparison	featurization
=	100
>	010
>=	110
<	001
<=	101
<>, !=	011

The featurization of `A > 5` could then look like this:

$$\underbrace{001}_{A}\ \underbrace{010}_{>}\ \underbrace{0.27}_{5}$$

To combine multiple selection predicates, there exist two approaches:

In **Singular Predicate Encoding**, as used in [35, 93], for a table with m attributes, the feature vector $\mathcal{F}$ has $4{\cdot}m$ entries. Since specific entries in the feature vector are reserved for each attribute, there is no need to encode the attribute id itself. For $m = 3$ and a query with predicates `A > 5 AND B = 7`, the query featurization looks like:

$$\overbrace{\underbrace{010}_{>}\ \underbrace{0.27}_{5}}^{A}\ \overbrace{\underbrace{100}_{=}\ \underbrace{0.15}_{7}}^{B}\ \overbrace{\underbrace{000}_{\text{no}}\ \underbrace{0.0}_{\text{pred.}}}^{\text{third attribute}}$$

Note that all entries are set to 0 for attributes for which the query contains no predicate. Further note that there can only be up to one predicate per attribute and there is no (good) way to support disjunctions, so all predicates must be connected by `AND`.

Predicate Set Convolution, as used in [47], builds on *deep sets* [96], which is a technique to featurize sets. First, each selection predicate is featurized using the above method. Then, all per-predicate featurizations are collected in a set P. Finally, a mapping from P to a feature vector $\mathcal{F}$ for the predicates is learned during training. The advantage of Predicate Set Convolution is that it supports multiple predicates per attribute. However, disjunctions are not supported. In addition, Predicate Set Convolution as a QFT does not allow for statements in terms of generalization.

9.2.1.2 Join Featurization

We now focus on query featurization for queries containing joins. In particular, this section discusses how to handle the tables and join predicates contained in some input query $\mathcal{Q}$. Again, there are two general approaches to featurize joins:

With **local models**, as used in [35,93,94], one model is built per sub-schema, i.e., either per base table or per join result. To estimate the result cardinality of some query, the selection predicates in the query are featurized and forwarded to the corresponding local models. The advantages are that (1) the size of the feature vector remains small and (2) when data in some tables change, only the models for sub-schemata containing this table have to be retrained. At first sight, a downside is the number of models, since there are $2^n - 1$ sub-schemata, i.e., combinations of n tables. However, in real applications, this number is reduced by relying on System R formulas [78,92] where models are built exactly for those sub-schemata for which the assumptions from [78], i.e., uniformity and independence assumptions, do not hold.

On the other hand, a **global model**, as used in [47, 50], represents a single estimator capable to estimate result cardinalities for all queries containing

arbitrary sub-schemata of n tables. In global models, the feature vector of a featurized query must also represent the accessed tables. Assuming that tables are joined following their key/foreign-key relationships, any QFT can be adapted to global models by appending a binary vector to the feature vector, where each entry corresponds to a specific table. For instance, for tables 1, 2, 3, and 4, the binary vector 1101 corresponds to a query where tables 1, 2, and 4 are joined (following their key/foreign-key relationships). For comparison, 0100 corresponds to a query on base table 2. In MSCN [47], the approach taken is slightly different. First, each table contained in the query is represented by a unique one-hot vector. Next, all one-hot vectors are collected in one set. Similarly, all join predicates are collected in a separate set. Then, exactly as described for Predicate Set Convolution, a mapping from sets to feature vectors is learned based on the deep sets technique [96].

9.2.2 Machine Learning Models

Now that we have seen how to obtain feature vectors, this section presents the basics of machine learning (ML). In addition, this section introduces two types of ML models, neural networks and gradient boosting, that are commonly used for cardinality estimation. In our evaluation in Section 9.5, we report on the accuracy of ML models under the different QFTs presented in Section 9.3.

In general, a ML model is a function $\widehat{f}$ that maps a feature vector $\mathcal{F} \in \mathbb{R}^d$, where d is the dimension of the feature space, to a *label* or *target* y from some target domain Y.

$$\widehat{f}\colon \mathbb{R}^d \to Y,\ \mathcal{F} \mapsto y, \tag{9.3}$$

where $\widehat{f}$ is an approximation for the relation $f \subseteq \mathbb{R}^d \times Y$. A characteristic of ML approaches is that $\widehat{f}$ is not defined explicitly but retrieved in a process called *training* or *learning*. In supervised learning, to which we restrict this section, the training to obtain $\widehat{f}$ is based on a *training set* $S_{\text{train}} = \{(\mathcal{F}_i, y_i)\} \subseteq f$. If the target Y is a continuous variable, like a selectivity, then training means to solve a *regression problem*, like in classical statistics. For the reminder of this section, we restrict ourselves to regression problems.

After training, $\widehat{f}$ can be applied to an arbitrary feature vector $\mathcal{F}$. Computing $\widehat{f}(\mathcal{F})$ is called a *forward pass*. When evaluating the accuracy of $\widehat{f}$ with either a *validation set* or *test set*[1] $S_{\text{eval}} = \{(\mathcal{F}_i, y_i)\} \subseteq f$, we usually observe that not all forward passes lead to the correct target value. Instead, in terms of the equation

$$\widehat{f}(\mathcal{F}_i) = y_i + \varepsilon_i \tag{9.4}$$

we observe that $\varepsilon_i \neq 0$ for some (or even all) $1 \leq i \leq |S_{\text{eval}}|$.

The errors ε_i have two sources: *reducible error* and *irreducible error* [89]. Reducible errors occur when $\widehat{f}$ does not perfectly describe f. Reducible errors can be reduced by improved training methods and larger training sets. We

[1] Validation sets are used for hyperparameter tuning, while test sets are used to evaluate model accuracy.

define that $\mathbb{R}^d$ fully depends on Y via f if f is a function $\mathbb{R}^d \to Y$ (instead of a relation). Irreducible errors occur if and only if Y does not fully depend on $\mathbb{R}^d$ via f. Then, we might observe the following in a training set $S_{\text{train}} \subseteq f$:

$$\exists (\mathcal{F}_1, y_1), (\mathcal{F}_2, y_2) \in S_{\text{train}} : \mathcal{F}_1 = \mathcal{F}_2 \nRightarrow y_1 = y_2, \tag{9.5}$$

i.e., determinism is violated, which generally reduces the accuracy of a ML model $\widehat{f}$. With respect to Equation 9.2 from the introduction, the problem from Equation 9.5 occurs for queries with different result cardinalities but equal feature vectors. The irreducible error cannot be reduced by improved training. However, it is not as irreducible as its name suggests. In Section 9.3, we come back to the irreducible error and present a desired property for query featurization techniques to reduce the irreducible error.

9.2.2.1 Neural Networks

Neural networks have been applied to various problems in DBMS [47, 49, 93]. Neural networks are able to solve supervised ML problems by modeling the dependencies within the problem as a black box. With increasing hardware performance over the last years, their training has become feasible on a large scale. Therefore, it seems to be evident that the database community relies on neural networks to solve the complex problem of cardinality estimation. Here, we focus on Multi-layer Perceptron or Feed-forward Networks (NN) and Multi Set Convolutional Networks (MSCN) because previous work has shown that these kinds of neural networks work best for queries with joins on complex schemata [47, 93]. In our evaluation in Section 9.5, we use the original network architecture from [47] and [93]. To show both the importance and versatility of query featurization, we extend [47, 93] to use our new QFTs.

9.2.2.2 Gradient Boosting

It has been observed that neural networks can be too complex and their training time takes too long. Hence, [28] proposes to use smaller and faster models, based on Gradient Boosting (GB). GB is a tree-based approach where weak learners are combined based on the residuals of a preceding learner. The GB estimator sums over P weak predictors $\widehat{F}_p$, each with weight λ_p, and adds a constant c.

$$\hat{f}(\mathcal{F}) = \sum_{p=1}^{P} \lambda_p \widehat{F}_p(\mathcal{F}) + c \tag{9.6}$$

In our case, each weak predictor $\widehat{F}_p$ is a simple decision tree. The general structure of GB models makes them very fast in training and forward passes. In our evaluation in Section 9.5, we show that GB trains and converges faster than the neural network approaches. Therefore, smaller training sets, and thus shorter training times, suffice to reach a given level of accuracy. Like with the neural network approaches, we combine GB with different QFTs and compare the performance of the QFT $\times$ ML model combinations.

All neural networks and GB models are input-agnostic, i.e., for a fixed input vector length, they can work with any numeric vector presented to them. This is very useful in the context of this work, since it allows us to vary the QFT without having to modify the models' architecture.

Finally, note that we also tested simpler models, like *linear regression* and *support vector regression*. However, we do not include these ML models in the further discussion and evaluation, since their estimates are worse by a significant factor.

9.3 Query Featurization

Query featurization is the process of encoding a query $\mathcal{Q}$ in a numerical vector, named *feature vector*. This feature vector then serves as input to a machine learning model.

This section presents three novel query featurization techniques (QFTs), where we focus on the encoding of selection predicates. The three novel QFTs are Range Predicate Encoding, Universal Conjunction Encoding, and Limited Disjunction Encoding. We restrict our scope to simple predicates, where an attribute value is compared to a literal using one of the comparison operators in $\{=, >, <, \geq, \leq, \neq\}$. We consider conjunctions as well as certain disjunctions of predicates. We also consider arbitrarily many predicates per attribute. Our QFTs can be applied to either a single table or, using the techniques presented in Section 9.2.1, to queries containing joins. Extensions to groupings and certain string predicates are discussed in Section 9.6.

The better a feature vector represents the predicates in a query, the better the query featurization technique. As part of our assessment of QFTs, we use the terms *information loss* and *lossless* from the field of data compression and apply the following definition.

Definition 9.3.1 (Result-lossless query featurization)**.** *Suppose we are given a database instance D. Let $S_{\mathcal{Q}} = \{\mathcal{Q}_i\}$ be a set of queries over D.*

Let g be a function that maps queries to feature vectors, i.e., a QFT. We say the feature vector $\mathcal{F}_i = g(\mathcal{Q}_i)$ is a result-lossless featurization of query $\mathcal{Q}_i$ if $\nexists \mathcal{Q}_j \in S_{\mathcal{Q}}, i \neq j : \mathcal{F}_i = \mathcal{F}_j \wedge R_i \neq R_j$, where R_i and R_j denote the results of query Q_i and Q_j, respectively.

If, for all $1 \leq i \leq |S_{\mathcal{Q}}|$, $\mathcal{F}_i$ is a result-lossless query featurization, then we call g a result-lossless QFT of $S_{\mathcal{Q}}$.

For instance, the query set $S_{\mathcal{Q}}$ might contain all range queries over some specific relation that exists in the database. Result-lossless query featurization means that all relevant information from a query is retained in its feature vector. Hence, as Figure 9.1 illustrates, no two queries with different results map to the same feature vector.

Note that the definition makes a statement about query results R_i rather than cardinalities $|R_i|$, which is the topic of this chapter. The motivation for the definition of result-lossless query featurization is its generality. To illustrate this

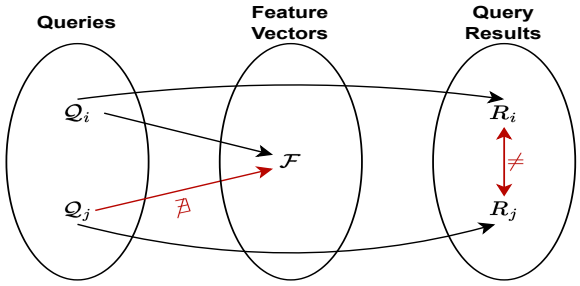


Figure 9.1: Illustration of result-lossless query featurization. When two queries have different results, their feature vector cannot be the same.

generality, suppose g is a result-lossless QFT for some query set $S_\mathcal{Q}$. Further suppose we are interested in some value $e(R_i)$, rather than the result R_i itself. For instance, $e(R_i)$ may be one of $|R_i|$, $|\pi^D_A(R_i)|$, or $|R_i \bowtie_{A=A} R_i|$, where A is an attribute of R_i. Then it is easy to see that g preserves for all $\mathcal{Q}_i \in S_\mathcal{Q}$ that $\nexists \mathcal{Q}_j \in S_\mathcal{Q}, i \neq j : \mathcal{F}_i = \mathcal{F}_j \wedge e(R_i) \neq e(R_j)$. This follows directly from the fact that $|\{R_i\}| \geq |\{e(R_i)\}|$ for all (deterministic) functions e. Hence, we say that g is also a e-lossless QFT. For the remainder of this chapter, we restrict our interest to *cardinality-lossless QFTs*, i.e., $e(R_i) = |R_i|$. With respect to the topic of this chapter, we refer to cardinality-lossless QFTs simply as *lossless QFTs*.

The following lemma captures a desired effect of lossless QFTs.

Lemma 9.3.1. *Given a database instance D, $S_\mathcal{Q} = \{\mathcal{Q}_i\}$, and $S_\mathcal{F} = \{\mathcal{F}_i\}$. Further suppose we are given a QFT $g : S_\mathcal{Q} \to S_\mathcal{F}$ and a function $h : S_\mathcal{Q} \to \mathbb{R}$, which returns the query result cardinality of a query, and the relation $f \subseteq \{(g(\mathcal{Q}), h(\mathcal{Q}) | \mathcal{Q} \in S_\mathcal{Q})\}$, which is approximated via $\widehat{f}$.*

If g is a lossless QFT, then $\widehat{f}$ has no irreducible errors.

Proof. By the definition of lossless QFT, $\forall \mathcal{Q}_i \in S_\mathcal{Q}$ it holds that

$$\nexists \mathcal{Q}_j \in S_\mathcal{Q}, i \neq j : g(\mathcal{Q}_i) = g(\mathcal{Q}_j) \wedge h(\mathcal{Q}_i) \neq h(\mathcal{Q}_j).$$

From this we must proof that $\widehat{f}$ has no irreducible errors, which holds if f is a function. We prove the lemma by contradiction. Suppose f is not a function. Then

$$\exists \mathcal{Q}_i, \mathcal{Q}_j \in S_\mathcal{Q}, i \neq j : g(\mathcal{Q}_i) = g(\mathcal{Q}_j) \wedge h(\mathcal{Q}_i) \neq h(\mathcal{Q}_j).$$

However, this is already contradicts the assumption that g is a lossless QFT. Hence, f must be a function. It follows that $\widehat{f}$ has no irreducible errors.

□

Note that a QFT is not either lossless or lossy. Instead, a QFT is lossless for a certain class of queries. When a QFT is not lossless for a class of queries, we say that information loss occurs, since an ML algorithm, to which feature

vectors serve as input, cannot distinguish between different input queries with the same feature vector representation.

The following shows that the previously discussed Singular Predicate Encoding is not a lossless QFT for queries with multiple predicates on some attribute A. Recall that Singular Predicate Encoding can represent only one predicate per attribute. Hence, for queries with up to one predicate on A, Singular Predicate Encoding is a lossless QFT. However, for queries with multiple predicates on A, the information about all but one predicate is lost. Thus, both selective queries with many predicates on attribute A and less selective queries with few predicates on A may induce the same feature vector. Hence, the result cardinality of a query does not fully depend on its feature vector.

9.3.1 Range Predicate Encoding

This section presents a straightforward but useful extension of Singular Predicate Encoding as discussed in Section 9.2.1. More evolved techniques are discussed in the next two sections. Range Predicate Encoding is a QFT that allows to encode, per attribute, either (1) a range predicate, where both open or closed ranges are supported, or (2) an equality predicate. In the existing work, range predicates were discussed in [94], but since their model was optimized for point queries, their cardinality estimate for queries containing range predicates is the sum of cardinality estimates for multiple point queries, which is computationally feasible only for small, discrete ranges. For large ranges, [94] employs a sampling technique.

Our predicate encoding technique builds on the observation that, in databases, all types of point and range predicates can be encoded to closed ranges. For instance, `A = 5` becomes $[5, 5]$ and `A` $\leq$ `5` becomes $[\min(A), 5]$. While the difference between range predicates including or excluding endpoints is often marginal, this can still be addressed: For integer attributes it is easy to see that `A < 5` corresponds to $[\min(A), 4]$ and for decimal attributes we can use a small step size, e.g. $[\min(A), 4.9]$. Since this is beneficial for some ML models, all ranges are normalized to $[0, 1]$ using the min and max values of each attribute.

The benefit of Range Predicate Encoding is that all queries with up to one equality, open range, or closed range predicate per attribute are featurized losslessly. Queries with multiple predicates per attribute are not supported. Neither are disjunctions, so all predicates must be connected by `AND`.

9.3.2 Universal Conjunction Encoding

The QFTs discussed thus far could be *read of* directly from the query but share one common caveat: Only a limited number of predicates can be encoded in the feature vector without information loss. The problem is inherent to the previous featurization techniques: SQL queries are of arbitrary length, but feature vectors have a fixed length!

Universal Conjunction Encoding builds on the observation that both the number of attributes in the data set, denoted by m, and the domain of each

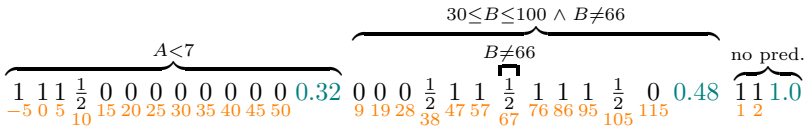
attribute remains constant. Hence, while the same attribute may occur in multiple predicates, we use the fact that no query references more than m distinct attributes. The idea that follows is to (1) partition the domain of each attribute, (2) give each partition one entry in the feature vector, and (3) assign a value to each entry that indicates whether the partition it represents satisfies the predicates in query $\mathcal{Q}$. This technique allows us to encode queries with arbitrarily many simple predicates connected via `AND`. To implement this idea, we discretize the domain of each attribute A into $n_A = \min(n,\ |A|_d)$ partitions, where n denotes some maximum number of partitions per attribute, e.g., 64, and $|A|_d$ denotes the number of distinct values in A. The feature vector entry corresponding to $v \in A$ has zero-based index

$$\min\left(\left\lfloor \frac{val - \min(A)}{\max(A) - \min(A)} \cdot n_A \right\rfloor,\ n_A - 1\right).$$

Hence, the partition each entry represents consists of consecutive values. Each feature vector entry indicates whether the corresponding partition qualifies the predicates in $\mathcal{Q}$. We use 0 to indicate that no value qualifies, $\frac{1}{2}$ to indicate that some values qualify, and 1 to indicate that all values qualify. Note that the numbers 0, $\frac{1}{2}$, 1 denote categories; they do not have the meaning of real or decimal numbers. The concatenation of the per-attribute featurization yields the total feature vector.

Note that the optimal choice of the maximum number of partitions n depends on the frequency distribution of the values in the m attributes. In general, we observe that each partition covers $1/n$ of the domain of some attribute A. Hence, with n=32, each partition covers roughly 3% of an attribute's domain. In fact, our evaluation (Sec. 9.5.3) supports n=32 as a reasonable heuristic. For attributes with high skew, a larger n may be necessary. Observe that it is easy to extend our approach to choose an attribute-specific n. One could also apply sophisticated partitioning techniques from the field of histograms, like v-optimal [74] and q-optimal [63] partitioning.

Example: Let R be a table with numeric attributes `A`, `B`, and `C`. Suppose $\min(A)$=-9, $\max(A)$=50, $\min(B)$=0, $\max(B)$=115 and `C` contains only values in $\{1, 2\}$. Let n=12 be the maximum per-attribute feature vector length. Then, a query over R with predicates `A < 7 AND B >= 30 AND B <= 100 AND B <> 66` induces the feature vector shown below. To easily reconstruct the example, the orange-colored indices show the largest attribute value each feature vector entry refers to. Ignore the teal-colored entries for now:



With respect to `A < 7`, note that 7 maps to the fourth entry in the vector of A since, according to the above zero-based index formula, we have that $\min\left(\left\lfloor \frac{7-(-9)}{50-(-9)} \cdot 12 \right\rfloor,\ 12-1\right) = 3$ This fourth entry is set to $\frac{1}{2}$. All entries to

the left are 1 to indicate that values smaller than 7 qualify. Accordingly, all entries to the right, but within the bounds of A's vector, are 0. Since there is no predicate on attribute `C`, and its domain consists of only two values, `C`'s featurization is the all-one vector 11. To reproduce the rest of the example, handle the predicates on B accordingly.

Algorithm 6 describes the steps taken to featurize a query $\mathcal{Q}$. Ignore the teal-colored lines for now, they are discussed later. In line 1, we define a map $\mathcal{M}$ that associates each attribute A, where A is an attribute in the relation under consideration, to a vector with n_A entries (see above). Initially, all n_A entries in each mapped vector $\mathcal{M}_A$ are set to 1. Then, starting in line 2, for each predicate p in query $\mathcal{Q}$, p is decomposed into the attribute A it refers to, its comparison operator *op*, and the literal *val* to which A is compared. A and *val* are then used to compute the index *idx* of the vector entry corresponding to *val*. If the vector entry $\mathcal{M}_A[idx]$ is 1, it is set to $\frac{1}{2}$. In particular, $\mathcal{M}_A[idx]$ can only be decreased, e.g., once it is zero, it remains zero. Then in lines 7 to 17, depending on the comparison operator *op*, vector entries that correspond to attribute values which do not qualify p are set to 0. Hence, each predicate p sets specific entries to 0 (or $\frac{1}{2}$). This captures the property that further predicates in a conjunction can make a query only more selective. The final feature vector, returned in line 22, is the concatenation of all per-attribute vectors.

Strictly speaking, if multiple distinct values correspond to a single feature vector entry, Universal Conjunction Encoding is not a lossless QFT for queries with arbitrary conjunctions of simple predicates. However, the following lemma captures a convergence property that holds.

Lemma 9.3.2. *Let $\mathcal{A}$ be a set of attributes and $\mathcal{Q}$ a query with an arbitrary conjunction of simple predicates over a subset of $\mathcal{A}$. Denote by $\mathcal{F}_n$ the feature vector as produced by Algorithm 6 for query $\mathcal{Q}$, where n denotes the maximum number of feature vector entries per attribute in $\mathcal{A}$.*

Then, the sequence $(\mathcal{F}_n)_{n\in\mathbb{N}}$ converges to a lossless query featurization (cf. Definition 9.3.1).

Proof. We proof the lemma by showing that $(\mathcal{F}_n)_{n\in\mathbb{N}}$ is an eventual constant sequence, for which convergence is known.

In general, a sequence $(a_i)_{i\in\mathbb{N}}$ is a function $\mathbb{N} \to X$ that maps natural numbers to objects from the codomain X. An eventual constant sequence is one where there exists $n_0 \in \mathbb{N}$ such that $a_n = a_{n_0}, \forall n > n_0, n \in \mathbb{N}$. By the definition of Universal Conjunction Encoding, $\mathcal{F}_n$ contains $n_A = \min(n,\ |A|_d)$ entries for each $A \in \mathcal{A}$, where $|A|_d$ denotes the number of distinct attributes in A. Let $n_{dvmax} := \max(\{|A|_d : A \in \mathcal{A}\})$. Clearly, for all $n > n_{dvmax}$, we have that $\mathcal{F}_n = \mathcal{F}_{n_{dvmax}}$. Hence, the sequence $(\mathcal{F}_n)_{n\in\mathbb{N}}$ is eventually constant. By [34, p. 42], all eventual constant sequences converge.

□

Hence, for large feature vectors where each entry corresponds only to one distinct value of an attribute, Universal Conjunction Encoding is a lossless query featurization for all queries with arbitrary conjunctions of simple predicates. For

Algorithm 6 Query featurization using Universal Conjunction Encoding

FeaturizeArbitraryConjunction($\mathcal{Q}$)

1 Let $\mathcal{M}$ map attributes to all-one per-attribute vectors
2 **for** $p \in$ Predicates($\mathcal{Q}$)
3 $A,\ op,\ val$ = Decompose(p)
4 n_A = GetEntryCount(A)
5 $idx = \min\left(\left\lfloor \frac{val-\min(A)}{\max(A)-\min(A)} \cdot n_A \right\rfloor,\ n_A-1\right)$
6 **if** $\mathcal{M}_A[idx] == 1$: $\mathcal{M}_A[idx] = \frac{1}{2}$
7 **if** $op \in$ {"=", "is"}
8 $\mathcal{M}_A[!idx] = 0$
9 **elseif** $op \in$ {">", ">="}
10 $\mathcal{M}_A[0 : idx] = 0$
11 $min_A = \max(min_A, val)$ // initially $\min(A)$
12 **elseif** $op \in$ {"<", "<="}
13 $\mathcal{M}_A[idx+1 : \text{Len}(\mathcal{M}_A)] = 0$
14 $max_A = \min(max_A, val)$ // initially $\max(A)$
15 **elseif** $op \in$ {"!=", "<>"}
16 // do nothing
17 $nots_A \cup = val$ // initially empty set
18 **for** $A \in$ Attributes($\mathcal{M}$)
19 c_A = CountBetween($nots_A, max_A, min_A$)
20 $r_A = \max(max_A - min_A - c_A,\ 0)$
21 Append($\mathcal{M}_A,\ r_A\ /\ (\max(A) - \min(A)+1)$
22 **return** Concat_all($\mathcal{M}$) // feature vector $\mathcal{F}$

smaller feature vectors, this featurization loses only information up to the size of the partition that a feature vector entry represents.

So far, we did not discuss the teal lines in Algorithm 6. Here, per-attribute selectivity estimates are appended to the feature vector. The per-attribute selectivity estimate of some attribute A is the ratio of distinct values from all distinct values in A's domain that qualify the selection predicates on A. A is assumed to be an integer attribute. In the teal lines of Algorithm 6, this estimate is obtained as the size of A's domain that qualifies r_A divided by the total size of A's domain $\max(A) - \min(A) + 1$. This corresponds to an estimate under uniformity assumption like in [78]. The model benefits when the partitions in the feature vector are coarse-grained or when trained on a few queries.

Note that for attributes with sufficiently small domain sizes, one feature vector entry corresponds to one distinct value of some attribute. In our implementation of Algorithm 6, we recognize this case and set entries only to 0 or 1 (but not $\frac{1}{2}$). For brevity, we omitted this aspect in Algorithm 6.

Further note that only line 6 of Algorithm 6 sets feature vector entries to $\frac{1}{2}$. In particular, for predicates of the form `A` $\leq$ `c` or `A` $\geq$ `c`, the partition corresponding to `c` is always set to $\frac{1}{2}$, even in cases where the partition fully qualifies. Similarly, for predicates of the form `A` $<$ `c` or `A` $>$ `c`, the partition

corresponding to c might not qualify at all but is still set to $\frac{1}{2}$. To improve this behavior, line 6 would need to be adjusted. A possible extension would then also be to store in each entry the ratio of qualifying distinct values from the corresponding partition, instead of only 0, $\frac{1}{2}$, or 1. For instance, if the partition of an entry $\mathcal{M}_A[i]$ corresponds to the values from 5 to 10 of integer attribute `A`, then a predicate `A` $\leq$ `8`, would set $\mathcal{M}_A[i] = \frac{8-5+1}{10-5+1} = \frac{2}{3}$.

9.3.3 Limited Disjunction Encoding

To the best of our knowledge, Limited Disjunction Encoding is the first QFT that is designed to take both conjunctions and disjunctions, i.e. predicates connected by `AND` as well as `OR`, into account. Limited Disjunction Encoding is essentially a generalization of Universal Conjunction Encoding. Note that others [94] have mentioned disjunctions, but only referred to the inclusion-exclusion principle.

Before we present Limited Disjunction Encoding, we address its limitations - the name already suggests that we cannot handle arbitrary disjunctions. We restrict ourselves to the following class of queries:

Definition 9.3.2 (Mixed query)**.** *A **compound predicate** P_A for some attribute A is an arbitrary combination (`AND`/`OR`) of arbitrarily many simple predicates on A. A **mixed query** is a conjunction of the compound predicates for an arbitrary subset of attributes.*

For example, the following mixed query asks for orders in the TPC-H dataset from either 1994 or 1996, with July 4th excluded in both years, that are either in progress (P) or finished (F) and with a price range from 1000 to 2000:

```
SELECT * FROM Orders WHERE
(o_orderdate >= '1994-01' AND o_orderdate <= '1994-12'
   AND o_orderdate <> '1994-07-04'
      OR
 o_orderdate >= '1996-01' AND o_orderdate <= '1996-12'
   AND o_orderdate <> '1996-07-04') AND
(o_orderstatus = 'P' OR o_orderstatus = 'F') AND
(o_totalprice > 1000 AND o_totalprice < 2000);
```

Note that the query contains three compound predicates, each enclosed by parentheses. Our example, as well as the original TPC-H and TPC-DS queries, illustrates disjunctions with both categorical data and ordinal data. Note that mixed queries do not have to follow a CNF or DNF form.

As for this class of queries, disjunctions occur only locally, i.e., per attribute, it is sufficient for our approach to address them at this level. The key idea of Limited Disjunction Encoding is to regard each conjunction in each compound predicate as a query that can be featurized using Universal Conjunction Encoding. Then, for each compound predicate, the per-conjunction featurizations can be merged by taking the entry-wise max over all per-conjunction featurizations. This merging technique captures the property that additional disjunctions make

queries only less selective. As in the previous section, the final feature vector is the concatenation of all per-attribute vectors.

Example: Recall table R from the example in the previous section. We featurize the mixed query

```
SELECT * FROM R WHERE
(A > -2 AND A <= 30 AND A != 7 OR A >= 42) AND B >= 39.5.
```

The compound predicate on A consists of two conjunctions. For each conjunction, a featurized representation is generated using Universal Conjunction Encoding. In particular, `A > -2 AND A <= 30 AND A != 7` is featurized to

$$\underbrace{0\tfrac{1}{2}1\tfrac{1}{2}111\tfrac{1}{2}0000}_{-2<A\leq 30\wedge A\neq 7}$$

and `A >= 42` is featurized to

$$\underbrace{0000000000\tfrac{1}{2}1}_{A\geq 42}.$$

The above per-conjunction vectors must then be merged by taking the entry-wise max, i.e.,

$$\underbrace{0\tfrac{1}{2}1\tfrac{1}{2}111\tfrac{1}{2}00\tfrac{1}{2}1}_{-2<A\leq 30\wedge A\neq 7\vee A\geq 42}$$

The compound predicate on B only consists of `B >= 39.5`, which is regarded as one conjunction. The featurized representation is

$$\underbrace{0000\tfrac{1}{2}11111111}_{B\geq 39.5}.$$

As in the example from the previous section, since attribute `C` is not mentioned in the query, its featurization is the all-one vector 11. The concatenation of the per-attribute vectors gives the final feature vector for the above mixed query:

$$\underbrace{0\tfrac{1}{2}1\tfrac{1}{2}111\tfrac{1}{2}00\tfrac{1}{2}1}_{-2<A\leq 30\wedge A\neq 7\vee A\geq 42}\overbrace{0000\tfrac{1}{2}11111111}^{B\geq 39.5}\underbrace{11}_{\text{no pred.}}.$$

For the sake of brevity, we do not include per-attribute selectivity estimates, which were illustrated in teal in the previous section.

Algorithm 7 outlines the implementation of Limited Disjunction Encoding. As before, the input is query $\mathcal{Q}$, and we start with a map from attributes to vectors. Then, for each compound predicate cp, a vector with all entries set to zero is created. The subroutine $\textsc{Attr}(cp)$ returns the attribute of cp, and V contains $n_{\textsc{Attr}(cp)}$ many entries. Recall that cp is a disjunction of multiple conjunctions. Starting in line 4, each conjunction is regarded as a query that serves as input to Algorithm 6, which returns a feature vector f. Then, to capture the property that further disjunctions make queries only less selective, V is merged

Algorithm 7 Query featurization using Limited Disjunction Encoding

```
FeaturizeLimitedDisjunction(Q)
1  Let M map attributes to all-one per-attribute vectors
2  for cp ∈ CompoundPredicates(Q)
3      Let V be an all-zero feat. vec. for attr(cp)
4      for d ∈ Split(cp, "OR")
5          f = FeaturizeArbitraryConjunction(d)
6          V = entrywise_max(V, f)
7      M[Attr(cp)] = V
8  return Concat_all(M)
```

by taking the entry-wise maximum for each entry in V and its corresponding entry in f. For simplicity, assume that the subroutine ENTRYWISE_MAX knows which entries in f refer to V. The final per-attribute V is stored in $\mathcal{M}$. As before, the concatenation of all per-attribute vectors gives the final feature vector for input query $\mathcal{Q}$.

Since merging feature vectors as in line 6 of Algorithm 7 directly resembles the semantics of `OR`, and the feature vectors to be merged converge to a lossless feature vector by Lemma 9.3.2, it follows that Limited Disjunction Encoding converges to a lossless query featurization of mixed queries.

9.4 Implementation

This section outlines how we adopt different ML models to use our QFTs. In our implementation, the code of the QFTs is independent of the ML model. Hence, each ML model may use any QFT. As discussed in Section 9.2.2, we focus on NN, MSCN, and GB models.

9.4.1 Local Models

The local model approach can be implemented with both NN and GB since local models operate on sub-schemata and, therefore, can be any arbitrary ML model [93]. We decide to use the original NN from [93] and a modified version where each neural network is replaced by a GB model. Of course, we adjust the necessary hyperparameters.

Extending NN and GB to use different QFTs is straightforward. The model's architecture remains as is and only the feature vectors presented to a model change. To achieve this, the code pipeline is adjusted so that queries flow through the configured QFT routine.

9.4.2 Global Models

As a common representative for global models in cardinality estimation, we extended the MSCN [45] implementation to use our new QFTs. Since in MSCN query featurization is intertwined with the model, we briefly outline our adjustments.

To featurize a query, MSCN splits the components of the query and collects them in three sets of feature vectors. The three sets of feature vector are: (1) a set of featurized join predicates, as described in 9.2.1.2, (2) a set of pairs, each consisting of a featurized table name, simply a one hot vector, together with a base table sample vector (details below), and (3) a set of featurized selection predicates. We briefly detail on the base table sample vector. First, note that MSCN is not a purely synopses based estimator. Instead, if available, MSCN additionally relies on base table samples. For a base table sample $T_i' \subseteq T_i$ of table T_i, the corresponding base table sample vector has length $|T_i'|$, where each entry $T_{i,j}'$ is either 0 or 1. If the jth sample qualifies the selection predicates from the query that refer to T_i, then $T_{i,j}' = 1$, otherwise $T_{i,j}' = 0$.

Our QFTs address the featurization of the selection predicates. The other two vector sets remain untouched by our implementation. To retain the set logic of MSCN, we featurize all selection predicates referencing the same attribute into one feature vector. Each per-attribute feature vector is then labeled by the attribute id and collected in a set. This set serves as the set of featurized selection predicates for MSCN, exactly as in the original implementation [45].

9.5 Evaluation

This section presents our experimental evaluation. We evaluate all QFT $\times$ ML model combinations. In the evaluation, we use two different data sets under different query workloads. We evaluate estimation accuracy under different scenarios, take memory consumption into account, and compare our best QFT $\times$ ML model combination to other established cardinality estimators. In particular, we give empirical answers to the following questions:

- Which QFT leads to the best estimation accuracy?
- Does the number of attributes or selection predicates explain dispersion in estimation accuracy?
- Does query drift impact estimation accuracy?
- Does the QFT choice impact training convergence?
- What is the time & memory cost of QFTs and models?
- What QFT $\times$ ML model combination do we recommend?
- How does our favored QFT $\times$ ML model combination compare to established cardinality estimation techniques?

Data sets & query workloads. We use two real-world data sets together with corresponding query workloads. The first data set, forest cover type (forest) [56] is popular both in the machine learning and cardinality estimation community and contains more than 580k entries with 55 attributes. For forest, we generate a query workload with *conjunctive queries*, i.e., predicates connected by `AND`. We draw k, $1 \leq k \leq 55$ distinct attributes uniformly at random and randomly generate a closed range predicate for each. Additionally, we generate $l, 0 \leq l \leq 5$ non-equality ($\neq$) predicates, for each of the k chosen attributes, that exclude values from the aforementioned range, where l is drawn uniformly at random. For instance, one of the queries from our evaluation is:

```
SELECT count(*) FROM forest
WHERE A7 >= 160 AND A7 <= 225 AND
A8 >= 45 AND A8 <= 237 AND A8 <> 220 AND A8 <> 186
```

In addition, we generate a second query workload with mixed queries, in the sense of Definition 9.3.2. The generation is the same as for conjunctive queries, except that we repeat the generation for the per-attribute predicates between $m, 1 \leq m \leq 3$ times and concatenate them via `OR`. For an example see the query below Definition 9.3.2. For both conjunctive and mixed queries, we generated 100k training queries and another 25k test queries.

As a second data set, the Internet Movie Database (IMDb) [53] is used. IMDb contains data on more than 2.5 million movies with around 4 million actors from more than 135 years. For testing, we use JOB-light, a collection of 70 hand-written SQL queries containing joins from [47]. The JOB-light queries contain between 2 and 5 joins. The selection predicates are only conjunctions of 1 to 5 predicates on 1 to 4 different attributes. The queries contain at most one range per attribute. For training, 231k generated training queries are used. In all query workloads, since training and test sets are disjoint, test set leakage is avoided.

Error metric. We use the q-error metric [66], defined as $\max(\frac{x}{e}, \frac{e}{x})$, to measure the deviation between a cardinality x and its estimate e. The q-error is a relative and symmetric metric. Essentially all relevant work on ML-based cardinality estimation employs the q-error [28, 35, 38, 47, 93, 94]. In our evaluation, we consider only queries with non-empty results, and all estimates are ≥ 1. Hence, we can ignore that the q-error is undefined for zero inputs. Note that the relative error $\frac{|e-x|}{x}$ is an insufficient metric for its systematic preference to estimators that underestimate [84].

Experimental setup. All QFTs are implemented in Python. For the NN, we use the Keras/TensorFlow implementation provided by the authors of [93]. The GB models are built with lightGBM. MSCN and its modifications are built upon the code published alongside the paper [45]. We ran all experiments on a computer with a AMD A10-7870K Radeon R7 CPU, where we did not use the integrated Radeon R7 GPU unit. The computer has 32 GB of main memory and a NVIDIA Tesla K20c GPU. The neural networks are trained only with the hyperparameters from their papers due to long training times. The GB models,

on the other hand, are trained with full hyperparameter tuning implying the presented results are based on the best configurations.

Abbreviations. Throughout this section, the following abbreviations serve as labels in plots:

simple	Singular Predicate Encoding
range	Range Predicate Encoding
conjunctive	Universal Conjunction Encoding
complex	Limited Disjunction Encoding
GB	Gradient Boosting
NN	Feed-Forward Network
MSCN	Multi Set Convolutional Network

Unless stated otherwise, Universal Conjunction Encoding and Limited Disjunction Encoding use 64 per-attribute entries each.

9.5.1 Quality under All QFT × Model Combinations

In this section, we analyze the estimation accuracy of all QFT × ML model combinations discussed in this chapter. Recall that the new QFTs presented in this chapter are Range Predicate Encoding (range), Universal Conjunction Encoding (conjunctive), and Limited Disjunction Encoding (complex). Singular Predicate Encoding (simple) serves as a benchmark comparison. The QFTs are combined with the ML models feed-forward neural network (NN), gradient boosting (GB), and Multi-Set Convolutional Network (MSCN).

Figure 9.2 shows the estimation errors observed for the forest data set. As usual, the bottom and top of a box are the 25% and 75% quantiles, and the middle band is the median. The lower and upper whiskers show the 1% and 99% percentiles, respectively. All boxplots refer to the conjunctive query workload, except for Limited Disjunction Encoding. Limited Disjunction Encoding refers to the mixed query workload, for which, to the best of our knowledge, no good comparison exists. To indicate this difference, we separate the plots of Limited Disjunction Encoding by a vertical line.

We note three things in Figure 9.2: (1) Given the Singular Predicate Encoding and Range Predicate Encoding QFT, the local model choice, GB or NN, does not make a significant difference. Note that for MSCN + simple, we show the original model from [47] but without a sample. Here, MSCN clearly outperforms GB + simple and NN + simple. (2) Given the Universal Conjunction Encoding and Limited Disjunction Encoding QFTs, GB and MSCN outperform NN. And (3), given the GB or MSCN model, Universal Conjunction Encoding and Limited Disjunction Encoding clearly outperform the other two QFTs.

In Figure 9.3, we explore the estimation accuracy for different numbers of attributes mentioned in the queries. We only show GB since NN underperforms GB over all number of attributes and MSCN performs worse than GB on joins queries, as will be shown later. Observe how the accuracy worsens for all QFTs in the number of attributes. Further observe that Universal Conjunction Encoding outperforms Singular Predicate Encoding and Range Predicate Encoding.

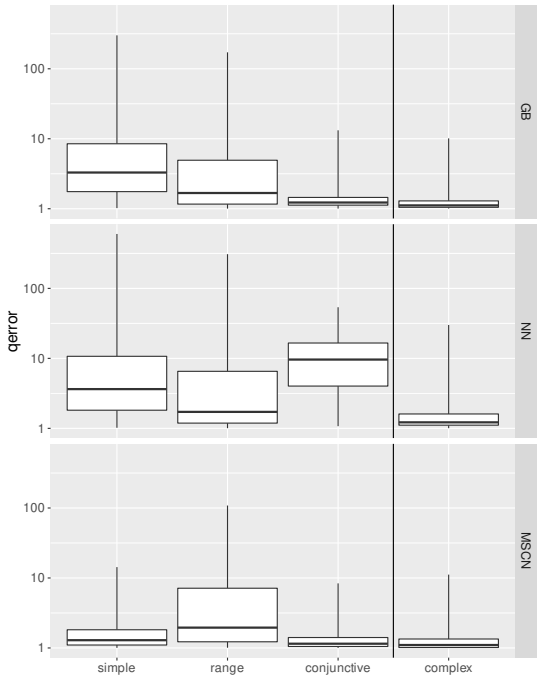


Figure 9.2: Error distribution by QFT × ML model combination. Dataset is forest. Limited Disjunction Encoding shows errors for mixed query workload. All other QFTs are for conjunctive query workload.

Note that while disjunctions are generally regarded as challenging in cardinality estimation, Limited Disjunction Encoding performs about as well as Universal Conjunction Encoding.

Figure 9.4 shows the estimation accuracy by number of predicates in the queries, again only for GB. Note that queries with two predicates refer to queries with a single range predicate, one predicate for the lower bound and the other one for the upper bound. Further, note that queries with three predicates refer to queries with one closed range predicate from which one value is excluded by a not-equal predicate. For its limited predicate encoding abilities, the observation that only Singular Predicate Encoding struggles with two predicates meets the expectation. Accordingly, as we go from two to three predicates, we observe a spike in the 99% error quantile (upper whisker) of Range Predicate Encoding. Observe that the accuracy further worsens as the number of predicates increases. Universal Conjunction Encoding and Limited Disjunction Encoding perform more consistently over varying numbers of predicates.

For queries containing joins, we report the JOB-light results in Table 9.1, which shows for each QFT × ML model combination the errors of the mean, median, 99% quantile, and maximum. Since JOB-light consists of only 70 queries,

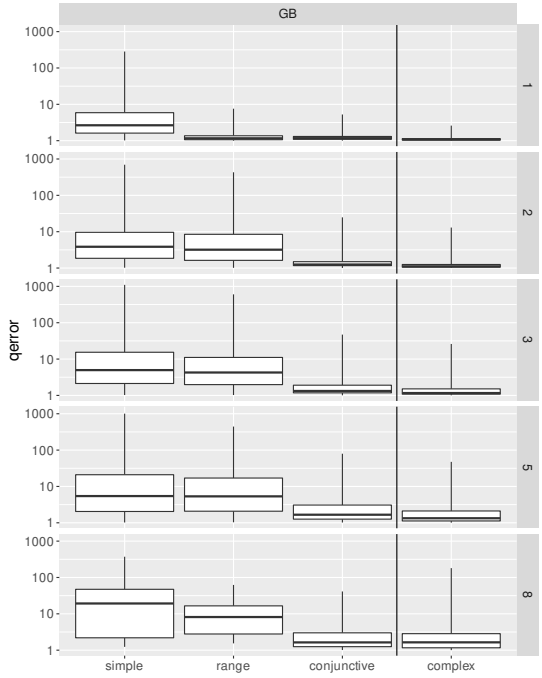


Figure 9.3: Estimation errors per QFT in the number of **attributes mentioned** in the queries.

the 99% quantile contains all errors except the worst outlier. For the feed-forward neural network model, Universal Conjunction Encoding significantly outperforms the other two QFTs. Overall, the estimates of GB + range are best. This comes as no surprise, since JOB-light queries contain at most one point predicate or range predicate per attribute. Universal Conjunction Encoding was configured to have 8 per-attribute entries for the NN and 32 per-attribute entries for GB. Limited Disjunction Encoding is not shown since JOB-light does not contain disjunctions and, hence, the feature vectors of Limited Disjunction Encoding and Universal Conjunction Encoding are equal.

Recall from Section 9.2.1.2 that for queries containing joins, there is a difference between local and global models. While Table 9.1 shows only local models, we address global models separately in Table 9.2 and focus on MSCN. *MSCN w/o mods* refers to the original MSCN from [47]. MSCN + conj is a modified version that uses our Universal Conjunction Encoding QFT. Observe that the errors in MSCN, both on average and over all quantiles, are significantly reduced with our QFT. In addition, note the gap between the accuracy of global and local models. Since the local model NN and the global model MSCN are both neural network approaches, we again show NN + conj in the last line of Table 9.2 to illustrate their difference. Observe that NN + conj has significantly

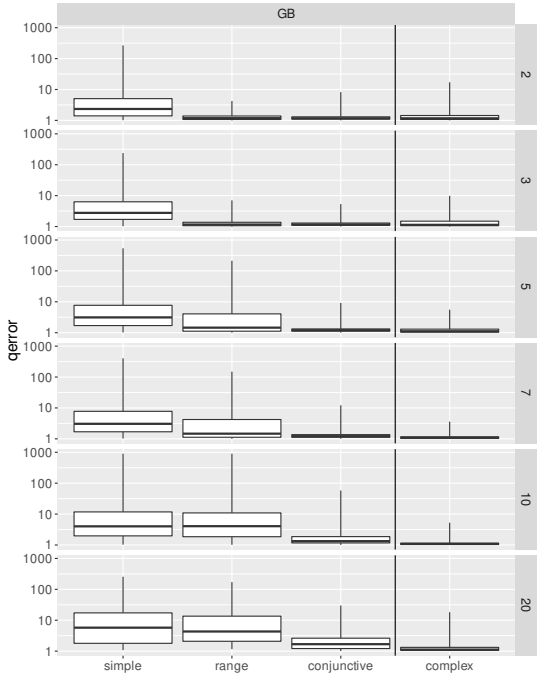


Figure 9.4: Estimation errors per QFT in the number of **predicates** in the queries.

lower errors than both MSCN w/o mods and MSCN + conj. Since the global model MSCN struggles with joins, we recommend to use local models.

Recall that, in Universal Conjunction Encoding and Limited Disjunction Encoding, we append per-attribute selectivity estimates to the feature vectors. In Table 9.3 we investigate the effect of these estimates. For multiple QFT $\times$ ML model combinations, we show results obtained with per-attribute selectivity estimates (w/ attrSel) and without per-attribute selectivity estimates (w/o attrSel). Note that in most cases the difference is marginal. However, the benefit is that in all cases except one, the worst case error (max) is reduced.

9.5.2 Comparison with Other Estimators

We identified GB + Universal Conjunction Encoding as the best estimator for conjunctive queries. For mixed queries (conjunctions and disjunctions), GB + Limited Disjunction Encoding is our best estimator. In this section, we compare our best query QFT $\times$ ML model combinations to the following estimators:

- *Postgres* is the cardinality estimator from PostgresSQL version 13.2., essentially independence assumption.

model + QFT	mean	median	99%	max
NN + simple	144.47	10.67	2507.34	3331.07
NN + range	110.23	7.60	2050.50	3573.30
NN + conj	19.97	5.74	129.45	134.37
GB + simple	4.03	1.88	34.06	56.39
GB + range	**3.92**	1.65	**29.77**	**45.51**
GB + conj	8.88	**1.52**	106.10	114.55

Table 9.1: 70 hand-written JOB-light join queries

model + QFT	mean	median	99%	max
MSCN w/o mods (global)	138.94	11.23	4209	5460
MSCN + conj (global)	119.83	5.26	1465	1811
NN + conj (local)	19.97	5.74	129	134

Table 9.2: JOB-light join queries: Local vs. Global Models

- *Sampling* is a 0.1% Bernoulli sample of the data. The sample is drawn independently per query.
- *Multi-set Convolutional Network (MSCN)* without modifications, where we did not use the optional sampling to solely judge the prediction accuracy of the ML model itself.

Recall that a detailed description of existing estimators is given in the related work, Section 4.

Figure 9.5 shows our best and the competing estimators for both the mixed and conjunctive query workload over the forest data set. In the plot, the query workload is partitioned by the number of attributes in each query. We discuss the *Conjunctive Queries* first. In the plot, note that all estimators lose accuracy with an increasing number of attributes. The accuracy of the Postgres estimator is worse compared to both ML approaches. For sampling, we observe a familiar phenomenon: It works in most cases, but has large tail errors. MSCN performs well, but the error varies a lot for different runs, as presented in [94]. Note, though, the difference between MSCN and our approach (*GB + conj*). In some

model	mean	median	99%	max
GB+conj w/ attrSel	2.65	1.12	20.19	4709.14
GB+conj w/o attrSel	2.93	1.23	25.78	3876.95
GB+comp w/ attrSel	2.95	1.11	18.31	6051.11
GB+comp w/o attrSel	2.92	1.06	16.00	8823.52
NN+conj w/ attrSel	3.65	1.36	19.80	23912.81
NN+conj w/o attrSel	4.00	1.28	16.93	38377.30
NN+comp w/ attrSel	5.08	1.21	37.54	16482.75
NN+comp w/o attrSel	39.74	3.20	268.39	246047.41

Table 9.3: Effect of per-attribute selectivity estimates

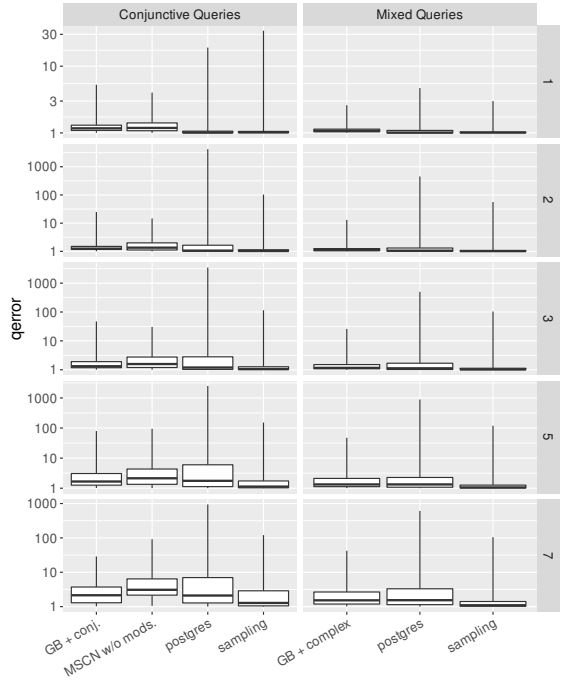


Figure 9.5: Our best QFT × ML model combinations (GB + conj./complex) are compared to established estimators. We consider both query workloads (conjunctive and mixed) for the forest data set. The figure shows the q-error distributions for different numbers of attributes in the queries.

cases, as specified by the number of attributes, we significantly outperform MSCN.

Looking at the *Mixed Queries*, we again note that all estimates worsen in the number of attributes mentioned in the queries. We observe that the Postgres estimator performs worst. For sampling, we again note bad tail errors. Our approach (*GB + complex*) has a slightly larger median and 75% error but significantly lower 99% errors than sampling. Since the standard implementation of MSCN does not support disjunctions, its performance cannot be demonstrated for this query workload.

9.5.3 Feature Vector Length

This section compares the accuracy of Universal Conjunction Encoding with different feature vector lengths and their respective memory consumption. In Section 9.3.2, we discussed that the number of per-attribute entries in the feature vector is a free parameter that can be varied. Table 9.4 shows the results for the JOB-light query workload for $\{8, 16, 32, 64, 256\}$ per-attribute entries.

no. entries	mean	median	99%	max
8	16.98	1.63	149.51	169.90
16	11.49	**1.52**	111.61	123.06
32	**8.88**	**1.52**	**106.10**	**114.55**
64	20.13	1.90	278.45	313.93
256	86.68	1.69	1347.91	1539.26

Table 9.4: Accuracy for different feature vector lengths

Note that one additional entry is always used for the per-attribute selectivity estimate (printed in teal in the illustration in Section 9.3.2). The ML model used is gradient boosting. We observe that, for the JOB-light workload, the best choice is 32 per-attribute entries for Universal Conjunction Encoding. When the number of entries is smaller than 32, information loss dominates, i.e., the feature vector is too far away from a lossless query featurization, thus causing larger errors. For more than 32 entries, the ML model struggles to learn the patterns encoded in the feature vector, which happens when the feature vector is too long, given the number of training queries.

9.5.4 Concept Drift

This section is concerned with how well eight QFT × ML model combinations generalize to unseen input/output pairs. In particular, *concept drift* describes the change of input and/or output characteristics. We make the following observation.

Key observation. *At any point in time, the data stored and/or queries executed in a DBMS may change abruptly and drastically.*

This observation does not apply to all machine learning tasks. For instance, cats in image recognition usually do not change their shape abruptly and drastically (evolution takes time).

9.5.4.1 Query Drift

This section presents an experiment where we simulate query drift. In this experiment, we use query workloads with different characteristics for training and testing. In particular, we use *low-dimensional* queries for training, i.e., queries mentioning at most two distinct attributes per query. For testing, we use *high-dimensional* queries, i.e., queries mentioning at least three distinct attributes per query. Here, the changed feature vectors are query drift for the ML model. In particular, the more attributes mentioned in the predicates of a query, the fewer entries are set to 1 in the query's feature vectors. In addition, the output characteristics, i.e., the query result sizes, change, which makes it hard for a model to generalize. The low-dimensional queries, used for training, have a mean query result size of 174 566 and 307 093 for the conjunctive and mixed workload, respectively. The high-dimensional queries, used for testing,

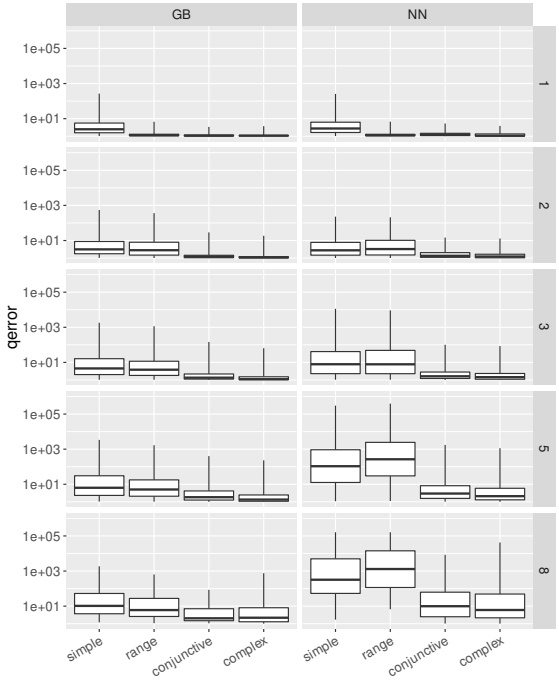


Figure 9.6: Training on queries with up to two attributes in query. Testing on queries with at least three attributes. Not all featurization/model combinations compensate this *query drift* well.

however, have mean query result sizes of only 79 805 and 131 376. Hence, in order to maintain a good estimation accuracy during testing, the model must, on average, produce estimates less than half as large as during training. Figure 9.6 shows the corresponding plots. The rows with 3, 5, and 8 attributes correspond to the test queries. The rows with 1 or 2 attributes correspond to the training queries, which we normally do not show, but are meant to illustrate the impact of the query drift. For GB, all featurizations generalize notably well. This can be observed by comparing Figure 9.6 to Figure 9.3, which shows the same aspects, but without query drift. When taking a closer look, we note that the 99% quantile error for the 8-attributes case is larger in Figure 9.6 than it was in Figure 9.3.

The right column of Figure 9.6 illustrates the case for the NN. For NN, a clear difference between low-dimensional training queries (≤ 2 attributes) and high-dimensional test queries (> 2 attributes) can be noted. The notable difference indicates that the NN overfits during training. In addition, the overfitting effect seems less severe for Limited Disjunction Encoding (complex) and Universal Conjunction Encoding (conjunctive) than for Range Predicate Encoding (range) and Singular Predicate Encoding (simple).

training queries	GB conj	comp	range	simple
10k	5.96	4.71	58.23	76.93
20k	4.31	4.11	56.07	63.98
30k	3.83	3.79	45.82	58.32
40k	3.43	3.83	43.74	54.23
50k	3.24	3.72	32.48	51.20
100k	2.93	2.96	32.50	47.29
training queries	**NN conj**	**comp**	**range**	**simple**
10k	28.44	17.91	283.20	386.20
20k	19.70	12.18	232.70	325.50
30k	13.15	10.44	98.17	267.80
40k	19.56	5.88	70.69	313.70
50k	8.32	4.45	57.37	149.02
100k	5.71	5.08	74.2	130.3

Table 9.5: Average estimation error for different model/size/featurization combinations. Forest data set.

9.5.4.2 Data Drift

Over time, the data stored in a database changes and all cardinality estimators become outdated. In ML, this is known as *data drift*. This section discusses how to react to data drift. We start by reporting measured run times for learning an estimator for 125k mixed queries for the forest data set. Note though that only 100k of the queries are used for training.

Generation/execution	≈3.5 days
Featurization	≈1.5 minutes
Training	GB: 6 sec, NN: 22 min, MSCN: 41min

Based on these numbers, we conclude that how to obtain queries and result cardinalities is critical. In Snowflake [81], queries and their result cardinalities can be derived from query logs or profiles. Featurization and training are rather cheap, in particular for estimators like GB. Since reinforcement learning is slow when looking at cumulated run times [71], we simply recommend to reconstruct models after data drift occurred. For deciding when to reconstruct, we recommend to follow Larson et al. [52], who propose to base the decision on query feedback.

9.5.5 Training Convergence

We report on how the average error changes in the number of training queries in Table 9.5. First, we observe that for all QFT × ML model combinations, the average error decreases, as the number of training queries increases. Next, note that the errors for NN are significantly larger than for GB. Finally, observe that, given the number of training queries, Universal Conjunction Encoding and

QFT	Simple	Range	Conj.	Comp.
μs per query	21.6	29.7	43.2	72.9

Table 9.6: Time consumption of QFTs for forest dataset

Limited Disjunction Encoding have significantly smaller average errors for both GB and NN than the other QFTs.

9.5.6 Time & Memory Consumption

In Table 9.6, we report on the time consumption of each QFT. The table shows the average time to featurize the forest workload queries. We note two things: (1) All QFTs run fast. All QFTs take significantly less than 100μs to featurize a query. (2) The time to featurize a query grows in the complexity of the QFT, with Singular Predicate Encoding being the fastest and Limited Disjunction Encoding being the slowest.

In our experiments, the average time each ML model takes to estimate the result cardinality of one query is 26μs for GB, 29μs for NN, and 33ms for MSCN. In addition, we list the estimation times reported in [28, 47, 94]. Dutt et al. observe an estimation time of around 100μs for GB and NN. For MSCN, Kipf et al [47] report estimation times of *"a few milliseconds"* while Yang et al [94] report a median estimation time of around 20ms. We suspect the factors that explain the differences are machine architecture, implementation details, and the queries considered.

We also report on the memory consumption of each estimator considered in this evaluation. Since Postgres relies on independence and uniformity assumptions, its estimator's memory footprint is negligible. For sampling, the size of a 0.1% sample depends on the data. For the forest dataset (142 MB in the Postgres DB), the sample consumes around 142 kB. For MSCN, we report the number of trainable parameters used for the selection predicates. In general, MSCN learns 8 linear transformations, implemented via `torch.nn.linear`, where the trainable parameters are the entries of a matrix and a bias vector. Two linear transformations are responsible for selection predicates and we only report their memory footprint. The bias vector of the two linear transformations has 256 entries each and the matrices have the dimensions $(55 + 4) \times 256$ and 256×256, resulting in a memory footprint of 320 kB for the trainable parameters, where we assumed only float32. Gradient boosting (GB) is the smallest estimator with only 4.8 kB memory consumption. Thus, we confirm results from [28], who observed that ML models can be small. The NN is the largest estimator with a footprint of more than 1 MB.

9.6 Discussion

This section discusses extensions and limitations of our approach to make it applicable for general-purpose cardinality estimation.

`GROUP BY` clauses. To the best of our knowledge, thus far, only Kipf et al. [46] covered `GROUP BY` clauses in ML-based cardinality estimation. However, `GROUP BY` clauses can significantly impact query result sizes. We outline how to featurize `GROUP BY` clauses such that combination with any QFT is easy. Suppose a binary vector with as many entries as attributes in the table under consideration, and we have defined some arbitrary mapping from attributes to vector entries. Then, this vector exactly describes the `GROUP BY` clause by setting the entry of each of the grouping attributes to 1. For instance, suppose we have the 5 attributes $A1$ to $A5$, then 01010 exactly corresponds to the clause `GROUP BY A2, A4`.

Note that cardinality estimation for queries containing aggregations, like `min/max/avg`, but no `GROUP BY` clause is simple: The query result size is always 1.

String predicates. Universal Conjunction Encoding and Limited Disjunction Encoding support string predicates of the form `C like a%`. Consider, for example, a column where all entries are strings of lower case letters, i.e., `[a-z]*`. With 26 entries in the per-attribute vector, each entry corresponds to the most significant letter of a word, e.g. words starting with `d` are represented by the fourth entry. For enhanced accuracy, more entries can be used, e.g. one can have 26*13 entries and the first entry corresponds to all words starting with `aa` and `ab`, but words starting with `ac` correspond to entry 2. Predicates like `C like %a`, however, are not supported.

Multi-dimensional histograms. Recall that for Universal Conjunction Encoding and Limited Disjunction Encoding, we partition each attribute domain. Then, each partition is represented by an entry in the feature vector. It should be noted that this is essentially the same as defining a bucketing scheme for a multi-dimensional histogram, where buckets correspond to rectangular partitions, as in [21]. Training using Universal Conjunction Encoding or Limited Disjunction Encoding then essentially means to learn the frequencies in each partition. The similarity between multi-dimensional histograms and ML-models for cardinality estimation was noted before [28].

9.7 Summary

We presented three new query featurization techniques (QFTs) and their impact on learning-based cardinality estimation. To the best of our knowledge, we are the first to consider queries containing both conjunctions and disjunctions of predicates for learning-based cardinality estimation. Previous QFTs either do not support disjunctions or only by rewriting to conjunctions. One key aspect of our work is to formally define desired properties of QFTs, i.e., lossless query featurization. Our QFTs are derived from this definition. The experimental evaluation indicates that using our QFTs leads to more accurate cardinality estimates. We find that our QFTs are robust since we examine the query workloads from different view points, like the number of predicates, attributes mentioned, and also query drift. In addition, our QFT $\times$ ML model combinations

compete well against established cardinality estimators. We demonstrated that our QFTs are model-independent by using them as a plug-in featurization layer for existing ML models. Hence, other researchers may choose to use our QFTs for their work on ML models.

Chapter 10

Conclusion and Outlook

In this thesis, we presented multiple novel cardinality estimation techniques. In the following, we summarize our contribution. The first contribution is CSE from Chapter 5, a novel approach to combine sampling with synopses for the purpose of estimating the selectivity of conjunctive queries. CSE proved to be a robust estimator with estimates at least as accurate as the estimates of the best competing estimator. Then, to the best of our knowledge, Chapter 6 presented the first detailed comparative analysis of intersection size estimation via AKMV sketches and HyperLogLog sketches. We found Binomial Mean Lookup to be a well-suited estimator for intersection size estimation. In Chapter 7, we presented a technique to estimate the size of a key/foreign-key join of two filtered relations. The key to derive a join size estimate is a memory-efficient data structure, the bucket sketch, which we have used to estimate the intersection size of two attribute sets after a selection predicate. Then, in Chapter 8, the focus shifted to multiple join queries. We presented an estimator that relies on two novelties. First, new operations for AKMV sketches. Second, translation grids, which track combinations of hashes from two AKMVs in a way that allows to approximate the joint existence distribution of two attribute sets. Finally, Chapter 9 presented three new query featurization techniques and their impact on learning-based cardinality estimation. To the best of our knowledge, we were the first to consider queries containing both conjunctions and disjunctions of predicates in the context of learning-based cardinality estimation. After testing queries with a varying number of predicates and attributes mentioned in a query, as well as varying result cardinalities, we found that our query featurization techniques are not just versatile, but also robust.

For future work, we see multiple directions. For CSE, it would be interesting to combine samples and synopses of base tables for the purpose of join selectivity estimation. Another possible direction of research in the area of join size estimation is the design of further methods for intersection size estimation and average multiplicity estimation. In particular, methods that are more versatile with respect to the selection and join predicates supported. For the multiple join size estimation approach using translation grids, it would be important to

solve the problem of having AKMVs with either zero tracked hashes or only tracked hashes with a multiplicity of zero. This problem relates to sampling-based approaches with zero qualifying samples. In terms of query featurization for machine learning models, we see research to support arbitrary disjunctions of predicates, instead of only certain disjunctions, as relevant future work. In addition, we see queries with nested sub-queries, arbitrary string predicates, or group-by clauses as a relevant research direction.

Bibliography

[1] A technical overview of Azure Cosmos DB. https://azure.microsoft.com/en-us/blog/a-technical-overview-of-azure-cosmos-db/. Accessed: 2022-06-24.

[2] Database SQL Tuning Guide: Histograms. https://docs.oracle.com/database/121/ TGSQL/tgsql_histo.htm. Accessed: 2018-02-15.

[3] IBM DB2: Runstats. ibm.com/support/knowledgecenter/SSEPEK_12.0.0/ugref/src/tpc/db2z_runstatssyntax.html. Accessed: 2020-08-27.

[4] IBM Knowledge Center: Managing DB2 performance. https://www.ibm.com/support/ knowledgecenter/en/SSEPEK_10.0.0/perf/src/tpc/ db2z_histogramstatistics.html. Accessed: 2018-02-15.

[5] SAP HANA SQL and System Views Reference: Create statistics. https://help.sap.com/viewer/4fe29514fd584807ac9f2a04f6754767/2.0.02/en-US/20d5252d7519101493f5e662a6cda4d4.html. Accessed: 2018-02-15.

[6] SLA for Azure Cosmos DB. https://azure.microsoft.com/en-us/support/legal/sla/cosmos-db/v1_4/. Accessed: 2022-06-24.

[7] Statistics in Microsoft's SQL Server and Azure SQL Database. https://docs.microsoft.com/en-us/sql/relational-databases/statistics/statistics. Accessed: 2018-02-15.

[8] The instacart online grocery shopping dataset 2017. instacart.com/datasets/grocery-shopping-2017. Accessed: 2020-02-05.

[9] The New and Improved Cardinality Estimator in SQL Server 2014. https://blogs.technet.microsoft.com/ dataplatforminsider/2014/03/17/the-new-and-improved-cardinality-estimator-in-sql-server-2014. Accessed: 2018-02-15.

[10] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *ACM SIGMOD*, 1999.

[11] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi. Mergeable summaries. *ACM Transactions on Database Systems (TODS)*, 38(4):1–28, 2013.

[12] G. Alefeld, F. A. Potra, and Y. Shi. Algorithm 748: enclosing zeros of continuous functions. *ACM Transactions on Mathematical Software (TOMS)*, 21(3):327–344, 1995.

[13] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. *Journal of Computer and System Sciences*, 64(3):719–747, 2002.

[14] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1):137–147, 1999.

[15] Z. Bar-Yossef, T. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer, 2002.

[16] R. Barber, E. Candes, A. Ramdas, and R. Tibshirani. Predictive inference with the jackknife+. *arXiv preprint arXiv:1905.02928*, 2019.

[17] K. Beyer, R. Gemulla, P. Haas, Reinwald, and Sismanis. Distinct-value synopses for multiset operations. *Communications of the ACM*, 2009.

[18] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *ACM SIGMOD*, 2007.

[19] R. P. Brent. An algorithm with guaranteed convergence for finding a zero of a function. *The Computer Journal*, 14(4):422–425, 1971.

[20] L. D. Brown, T. T. Cai, and A. DasGupta. Interval estimation for a binomial proportion. *Statistical science*, 16(2):101–133, 2001.

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

[22] W. Cai, M. Balazinska, and D. Suciu. Pessimistic cardinality estimation: Tighter upper bounds for intermediate join cardinalities. In *ACM SIGMOD*, 2019.

[23] Y. Chen and K. Yi. Two-level sampling for join size estimation. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 759–774. ACM, 2017.

[24] C. J. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934.

[25] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *Proceedings of the 31st international conference on Very large data bases*, pages 13–24. VLDB Endowment, 2005.

[26] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1–3):1–294, 2012.

[27] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[28] A. Dutt, C. Wang, A. Nazi, S. Kandula, V. Narasayya, and S. Chaudhuri. Selectivity estimation for range predicates using lightweight models. *Proceedings of the VLDB Endowment*, 12(9):1044–1057, 2019.

[29] O. Ertl. New cardinality estimation methods for hyperloglog sketches. *arXiv:1706.07290*, 2017.

[30] M. Ferdjallah. *Introduction to Digital Systems: Modeling, Synthesis, and Simulation Using VHDL*. John Wiley & Sons, 2011.

[31] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*, pages 137–156. Discrete Mathematics and Theoretical Computer Science, 2007.

[32] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[33] M. Freitag and T. Neumann. Every row counts: Combining sketches and sampling for accurate group-by result estimates. *ratio*, 1:1–39, 2019.

[34] E. Gaughan. *Introduction to analysis*, volume 1. American Mathematical Soc., 2009.

[35] S. Hasan, S. Thirumuruganathan, J. Augustine, N. Koudas, and G. Das. Deep learning models for selectivity estimation of multi-attribute queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1035–1050, 2020.

[36] D. Havenstein, P. Lysakovski, N. May, G. Moerkotte, and G. Steidl. Fast entropy maximization for selectivity estimation of conjunctive predicates on cpus and gpus. EDBT, 2020.

[37] A. Hertzschuch, G. Moerkotte, W. Lehner, N. May, F. Wolf, and L. Fricke. Small selectivities matter: Lifting the burden of empty samples. In *Proceedings of the 2021 International Conference on Management of Data*, pages 697–709, 2021.

[38] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig. Deepdb: learn from data, not from queries! *Proceedings of the VLDB Endowment*, 13(7):992–1005, 2020.

[39] D. Huang, D. Y. Yoon, S. Pettie, and B. Mozafari. Joins on samples: A theoretical guide for practitioners. *arXiv preprint arXiv:1912.03443*, 2019.

[40] Y. E. Ioannidis and S. Christodoulakis. On the propagation of errors in the size of join results. In *ACM SIGMOD*, 1991.

[41] Y. Izenov, A. Datta, F. Rusu, and J. H. Shin. Compass: Online sketch-based query optimization for in-memory databases. In *Proceedings of the 2021 International Conference on Management of Data*, pages 804–816, 2021.

[42] Z. Karnin, K. Lang, and E. Liberty. Optimal quantile approximation in streams. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 71–78. IEEE, 2016.

[43] A. Kasperski. *Discrete optimization with interval data*. Springer, 2008.

[44] F. Kastrati and G. Moerkotte. Optimization of conjunctive predicates for main memory column stores. *Proceedings of the VLDB Endowment*, 9(12):1125–1136, 2016.

[45] A. Kipf. Learned Cardinalities in PyTorch, 2019. Accessed: 2022-05-02.

[46] A. Kipf, M. Freitag, D. Vorona, P. Boncz, T. Neumann, and A. Kemper. Estimating filtered group-by queries is hard: Deep learning to the rescue. In *1st International Workshop on Applied AI for Database Systems and Applications*, 2019.

[47] A. Kipf, T. Kipf, Radke, Leis, Boncz, and Kemper. Learned cardinalities: Estimating correlated joins with deep learning. *CIDR*, 2019.

[48] R. P. Kooi. The optimization of queries in relational databases. 1981.

[49] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504, 2018.

[50] S. Lakshmi and S. Zhou. Selectivity estimation in extensible databases-a neural network approach. In *VLDB*, volume 98, pages 24–27, 1998.

[51] R. J. Larsen, M. L. Marx, et al. *An introduction to mathematical statistics and its applications*, volume 5. Pearson, 2017.

[52] P.-A. Larson, W. Lehner, J. Zhou, and P. Zabback. Cardinality estimation using sample views with quality assurance. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 175–186, 2007.

[53] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *Proceedings of the VLDB Endowment*, 9(3):204–215, 2015.

[54] V. Leis, B. Radke, A. Gubichev, A. Kemper, and T. Neumann. Cardinality estimation done right: Index-based join sampling.

[55] V. Leis, B. Radke, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. Query optimization through the looking glass, and what we found running the join order benchmark. *VLDB*, 2018.

[56] M. Lichman. UCI machine learning repository, 2013.

[57] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. *ACM SIGMOD Record*, 27(2):426–435, 1998.

[58] V. Markl, P. J. Haas, M. Kutsch, N. Megiddo, U. Srivastava, and T. M. Tran. Consistent selectivity estimation via maximum entropy. *The VLDB Journal—The International Journal on Very Large Data Bases*, 16(1):55–76, 2007.

[59] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent. *Advances in neural information processing systems*, 12, 1999.

[60] M. Menne, I. Durre, B. Korzeniewski, S. McNeal, K. Thomas, X. Yin, S. Anthony, R. Ray, R. Vose, B. Gleason, et al. Global historical climatology network-daily (ghcn-daily), version 3. *NOAA National Climatic Data Center*, 2012.

[61] M. J. Menne, I. Durre, R. S. Vose, B. E. Gleason, and T. G. Houston. An overview of the global historical climatology network-daily database. *Journal of Atmospheric and Oceanic Technology*, 29(7):897–910, 2012.

[62] G. Moerkotte. Building query compilers. *available at db. informatik. uni-mannheim. de/moerkotte. html. en*, 2006.

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

[64] G. Moerkotte and A. Hertzschuch. alpha to omega: the g (r) eek alphabet of sampling. CIDR, 2020.

[65] G. Moerkotte, M. Montag, A. Repetti, and G. Steidl. Proximal operator of quotient functions with application to a feasibility problem in query optimization. *Journal of computational and applied mathematics*, 285:243–255, 2015.

[66] G. Moerkotte, T. Neumann, and G. Steidl. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proceedings of the VLDB Endowment*, 2(1):982–993, 2009.

[67] S. Molini. Optimization technique for dealing with data skew on foreign key joins, Dec. 13 2011. US Patent 8,078,610.

[68] A. Nazi, B. Ding, V. Narasayya, and S. Chaudhuri. Efficient estimation of inclusion coefficient using hyperloglog sketches. *VLDB*, 2018.

[69] R. G. Newcombe. Two-sided confidence intervals for the single proportion: comparison of seven methods. *Statistics in medicine*, 17(8):857–872, 1998.

[70] J. Nocedal and S. Wright. Numerical optimization, series in operations research and financial engineering. *Springer, New York, USA, 2006*, 2006.

[71] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi. An empirical analysis of deep learning for cardinality estimation. *arXiv preprint arXiv:1905.06425*, 2019.

[72] G. Piatetsky-Shapiro and C. Connell. *ACM Sigmod Record*, 1984.

[73] H. Pirahesh, J. M. Hellerstein, and W. Hasan. Extensible/rule based query rewrite optimization in starburst. *ACM Sigmod Record*, 21(2):39–48, 1992.

[74] V. Poosala, P. Haas, Ioannidis, and Shekita. Improved histograms for selectivity estimation of range predicates. *ACM SIGMOD*, 1996.

[75] F. Rusu and A. Dobra. Sketches for size of join estimation. *ACM Transactions on Database Systems (TODS)*, 33(3):1–46, 2008.

[76] H. Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[77] S. L. Schulz. Combining histograms and hyperloglog sketches for distinct value estimation, 2020.

[78] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34. ACM, 1979.

[79] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Readings in Artificial Intelligence and Databases*, pages 511–522. Elsevier, 1989.

[80] M. Shekelyan, A. Dignös, and J. Gamper. Digithist: a histogram-based data summary with tight error bounds. *Proceedings of the VLDB Endowment*, 10(11):1514–1525, 2017.

[81] Snowflake. Understanding Your Snowflake Utilization, Part 1: Warehouse Profiling, 2017. Accessed: 2021-11-11.

[82] S. K. Thompson. *Estimating Proportions, Ratios, and Subpopulation Means*, pages 57–66. John Wiley and Sons, Inc., 2012.

[83] S. K. Thompson. *Sampling*, volume 755. John Wiley & Sons, 2012.

[84] C. Tofallis. A better measure of relative prediction accuracy for model selection and model estimation. *Journal of the Operational Research Society*, 66(8):1352–1362, 2015.

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

[86] D. Vengerov, A. C. Menck, M. Zait, and S. P. Chakkappen. Join size estimation subject to filter conditions. *VLDB*, 2015.

[87] A. Wächter and L. Biegler. IPOPT-an interior point OPTimizer, 2009.

[88] T. Wang and C.-Y. Chan. Improved correlated sampling for join size estimation. In *IEEE ICDE*, 2020.

[89] W. Wheeler. Reducible vs irreducible error, 2018. Accessed: 2022-05-14.

[90] E. B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[91] F. Wolf, N. May, P. R. Willems, and K.-U. Sattler. On the calculation of optimality ranges for relational query execution plans. In *Proceedings of the 2018 International Conference on Management of Data*, pages 663–675, 2018.

[92] L. Woltmann, C. Hartmann, D. Habich, and W. Lehner. Best of Both Worlds: Combining Traditional and Machine Learning Models for Cardinality Estimation. aiDM '20, New York, NY, USA, 2020. Association for Computing Machinery.

[93] L. Woltmann, C. Hartmann, M. Thiele, D. Habich, and W. Lehner. Cardinality estimation with local deep learning models. In *aiDM*, 2019.

[94] Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. Hellerstein, S. Krishnan, and I. Stoica. Deep unsupervised cardinality estimation. *VLDB*, 2019.

[95] X. Yu, N. Koudas, and C. Zuzarte. Hase: a hybrid approach to selectivity estimation for conjunctive predicates. In *International Conference on Extending Database Technology*, pages 460–477. Springer, 2006.

[96] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[97] Y. Zhou, Y. Zhou, M. Chen, and S. Chen. Persistent spread measurement for big network data based on register intersection. *POMACS*, 2017.