Benchmarking Approximate Consistent Query Answering

Benchmarking Approximate Consistent Query Answering

Marco Calautti
University of Trento
marco.calautti@unitn.it

Marco Console
University of Edinburgh
mconsole@inf.ed.ac.uk

Andreas Pieris
University of Edinburgh
apieris@inf.ed.ac.uk

ABSTRACT

Consistent query answering (CQA) aims to deliver meaningful answers when queries are evaluated over inconsistent databases. Such answers must be certainly true in all repairs, which are consistent databases whose difference from the inconsistent one is somehow minimal. Although CQA provides a clean framework for querying inconsistent databases, it is arguably more informative to compute the percentage of repairs in which a candidate answer is true, instead of simply saying that is true in all repairs, or is false in at least one repair. It should not be surprising, though, that computing this percentage is computationally hard. On the other hand, for practically relevant settings such as conjunctive queries and primary keys, there are data-efficient randomized approximation schemes for approximating this percentage. Our goal is to perform a thorough experimental evaluation and comparison of those approximation schemes. Our analysis provides new insights on which technique is indicated depending on key characteristics of the input.

ACM Reference Format:
Marco Calautti, Marco Console, and Andreas Pieris. 2020. Benchmarking Approximate Consistent Query Answering. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 29 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A database is inconsistent if it does not conform to its specifications given in the form of constraints. There is a consensus that inconsistency is a real-life phenomenon that arises due to many reasons such as integration of conflicting sources. With the aim of obtaining conceptually meaningful answers to queries posed over inconsistent databases, Arenas, Bertossi, and Chomicki introduced in the late 1990s the notion of Consistent Query Answering (CQA) [1]. The key elements underlying CQA are (1) the notion of *repair* of an inconsistent database D, that is, a consistent database whose difference with D is somehow minimal, and (2) the notion of query answering based on *certain answers*, i.e., answers that are entailed by every repair. Here is a simple example taken from [6]:

Example 1.1. Consider the schema $\mathbf{S}$ consisting of a single relation Employee(id, name, dept) that comes with the constraint that the first attribute, i.e., the id, is the key of the relation Employee. Consider also the database D over $\mathbf{S}$ consisting of the tuples:

$$(1, \text{Bob}, \text{HR}) \quad (1, \text{Bob}, \text{IT}) \quad (2, \text{Alice}, \text{IT}) \quad (2, \text{Tim}, \text{IT}).$$

Observe that the above database is inconsistent w.r.t. the key constraint since we are uncertain about Bob's department, and the name of the employee with id 2. In this case, to devise a repair, we only need to keep one of the two atoms that are in a conflict. In this way, we obtain a $\subseteq$-maximal subset of D that is consistent. Consider now the Boolean query that asks whether employees 1 and 2 work in the same department. This query is true only in two repairs, and thus, according to certain answers, is not entailed. ■

CQA has been extensively studied both from a theoretical point of view (see, e.g., [17–20, 25]), and a practical point of view (see, e.g., [11, 13, 16]). Nevertheless, the CQA approach comes with a conceptual limitation. The notion of certain answers only says that a candidate answer is entailed by *all* repairs, or is not entailed by *at least one* repair. But, as it has been discussed in [6, 7], the former is too strict, while the latter is not very useful in a practical context.

A Refined Approach. With the aim of obtaining more informative answers, we would like to rely on a notion that is more flexible than certain answers that tells us how likely a candidate answer is to be a consistent answer. This is achieved via the notion of relative frequency, which essentially computes the percentage of repairs that entail a candidate answer [6]. To illustrate this notion, let us consider again Example 1.1. The relative frequency of the empty tuple (which is the only candidate answer since the query is Boolean) is 50% since, out of four repairs in total, only two satisfy the query. It should not be surprising, though, that the problem of computing the exact relative frequency of a candidate answer is computationally very hard; in fact, it is #P-hard in data complexity, i.e., when the query and the constraints are fixed, even if we focus on conjunctive queries and primary key constraints [21, 22].[1] With such a computationally hard problem in our hands, as is customary in the literature, the way to proceed is to give up exact solutions, and target data-efficient approximations with explicit error guarantees.

Data-efficient Approximations. For practically relevant settings, in particular, conjunctive queries and primary keys, which is the main concern of the present work, we can approximate the relative frequency f of a candidate answer via a data-efficient randomized approximation scheme, i.e., a randomized algorithm that runs in polynomial time in the size of the input database, that computes a number a with relative error guarantees [6, 10]. The latter means that $|a - f| \leq \epsilon \cdot f$, for a fixed $\epsilon > 0$, or, in other words, the relative difference between f and a is bounded by ϵ. Furthermore, since a is the output of a randomized procedure, this should hold with a high probability, i.e., at least $1 - \delta$ for some small $0 < \delta < 1$.

[1] By primary keys we mean that each relation has at most one key, which is usually what we encounter in real-life database schemas.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The main data-efficient randomized approximation schemes for approximating the relative frequency of a candidate answer in the case of conjunctive queries and primary keys rely on ideas and techniques that have been originally proposed in the context of approximating the number of satisfying assignments of DNF Boolean formulas [15]. In fact, a wide range of problems such as network reliability [14], querying probabilistic databases [10], and, of course, querying inconsistent databases [6], to name a few, can benefit from the approximation schemes devised in the DNF context.

There are three main approaches, coming from the DNF literature, that one can exploit in order to devise a data-efficient randomized approximation scheme for our problem. Consider a database D, a set of primary keys Σ, a conjunctive query Q, and a candidate answer $\bar{t}$, i.e., a tuple of constant values from the domain of D:

▶ According to the first approach, which we call *natural*, a carefully chosen number N of repairs of D w.r.t. Σ are sampled uniformly at random, and the ratio S/N, with S being the number of positive samples, i.e., the repairs for which $\bar{t}$ is an answer to Q, provides a good approximation of the relative frequency of $\bar{t}$.

▶ The second approach, which we call *KL-based*, where KL stands for Karp and Luby, the authors that proposed this approach, is similar in spirit to the natural one with the crucial difference that the samples are now drawn from a slightly more sophisticated sampling space $\mathcal{S}$, which is sometimes called symbolic [24]. The goal here is to approximate the ratio $r/|\mathcal{S}|$, where r is the number of repairs of D w.r.t. Σ for which $\bar{t}$ is an answer to Q, which can be then converted into an approximation of the relative frequency of $\bar{t}$. The symbolic sampling space is useful in those cases where the number of repairs entailing the candidate answer is much smaller than the number of all repairs, i.e., the relative frequency is very small, and thus, approximating it directly via the natural approach will require a large number N of samples to obtain a good approximation. Consequently, $\mathcal{S}$ is designed in such a way that the ratio $r/|\mathcal{S}|$ is much larger than the relative frequency in the relevant cases.

▶ The third approach relies on an approximation scheme, originally proposed in the DNF context, for a problem known as the *union of sets* [15]: given the description of $n \geq 1$ sets $S_1, \ldots, S_n$, compute $|\bigcup_{i=1}^{n} S_i|$. It turned out that computing the relative frequency of $\bar{t}$ can be efficiently reduced to the union of sets problem, and thus inherit the corresponding approximation scheme.

It was generally known that techniques from the DNF literature can be adapted to our setting. Nevertheless, this is the first work that explicitly applies those techniques to the CQA setting.

Main Objective. While the refined approach to querying inconsistent databases, which exploits the more informative notion of relative frequency, has been adopted and studied in theory, and several data-efficient randomized approximation schemes can be devised for the central setting of conjunctive queries and primary keys, there is no corresponding infrastructure that can be used to experimentally evaluate such techniques. The main objective of this work is to take a major step in rectifying this state of affairs.

We present a benchmark (test infrastructure and test scenarios) for randomized approximation schemes that are designed to deal with conjunctive queries and primary keys, covering a wide range of scenarios. We have developed datasets of varying amount of inconsistency, and conjunctive queries of varying structural complexity by carefully tuning key static and dynamic query parameters that affect the performance of the approximation schemes. At this point, let us stress that the proposed benchmark, although it has been designed having approximation schemes in mind, it is quite generic, and can serve as the basis for evaluating algorithms that target the exact relative frequency, or certain answers. We then proceed to analyze the main data-efficient randomized approximation schemes for approximating the relative frequency of a candidate answer in the case of conjunctive queries and primary keys on our benchmark, with the aim of answering the following research questions:

(1) How does the performance of the approximation schemes in question is affected by the amount of inconsistency and the structural complexity of the conjunctive query?
(2) Can we arrive at definitive conclusions that can guide the choice of the randomized approximation scheme to be applied depending on some key characteristics of the input?
(3) Is, in the end, approximate consistent query answering as described above feasible in practice?

Main Outcome. Our experimental analysis revealed a striking difference between Boolean and non-Boolean conjunctive queries. In the former case, sampling from the natural space is the indicated approach, no matter the amount of inconsistency and the structural complexity of the query. This is not true for non-Boolean queries, where the approximation scheme that adopts the natural approach is the worst performer. Instead, the KL-based approach is the indicated one, no matter the amount of inconsistency and the structural complexity of the query. Finally, we can safely claim that approximate CQA in the presence of primary key constraints is feasible in practice. We have seen that for the modest scenarios from our benchmark, which is what we expect to face in practice, the runtime of the best performing approximation scheme is encouraging considering the hardness of the problem at hand.

Additional details on the approximation schemes, the implementation, the test infrastructure, and the test scenarios are deferred to the appendix. The actual implementation, as well as the actual test scenarios, can be found at https://gitlab.com/mcalautti/cqabench.

2 PRELIMINARIES

We recall the basics on relational databases, keys, and conjunctive queries. Given $n > 0$, we may write $[n]$ for the set $\{1, \ldots, n\}$.

Relational Databases. We assume a countably infinite set $\mathbf{C}$ of *constants* from which database elements are drawn. A *(relational) schema* $\mathbf{S}$ is a finite set of relation symbols (or predicates) with associated arity. We write R/n to denote that R has arity n. A *fact* over $\mathbf{S}$ is an expression of the form $R(c_1, \ldots, c_n)$, where $R/n \in \mathbf{S}$ with $n > 0$, and $c_i \in \mathbf{C}$ for each $i \in [n]$. A *database* over $\mathbf{S}$ is a finite set of facts over $\mathbf{S}$. The *active domain* of a database D, denoted $\mathrm{dom}(D)$, is the set of constants occurring in D.

Key Constraints. A *key constraint* (or simply *key*) κ over a schema $\mathbf{S}$ is an expression of the form $\mathrm{key}(R) = A$, where $R/n \in \mathbf{S}$ and $A \subseteq [n]$. Such an expression is called an R-key. Given an n-tuple of constants $\bar{t} = (c_1, \ldots, c_n)$, and a set $A = \{i_1, \ldots, i_m\} \subseteq [n]$, for some $m \in [n]$, we write $\bar{t}[A]$ for the tuple $(c_{i_1}, \ldots, c_{i_m})$, i.e., the projection of $\bar{t}$ over the positions in A. A database D satisfies κ if, for every two facts $R(\bar{t}), R(\bar{s}) \in D$, $\bar{t}[A] = \bar{s}[A]$ implies $\bar{t} = \bar{s}$. We

say that D is *consistent* w.r.t. a set Σ of keys, written $D \models \Sigma$, if D satisfies each key in Σ; otherwise, is *inconsistent* w.r.t. Σ. In this work, we focus on sets of *primary keys*, i.e., sets of keys that, for each predicate R of the underlying schema, have at most one R-key. For technical clarity, we assume, w.l.o.g., that each key constraint is of the form $\mathsf{key}(R) = \{1, \ldots, m\}$.

Conjunctive Queries. Let $\mathbf{V}$ be a countably infinite set of *variables* disjoint from $\mathbf{C}$. An atom over a schema $\mathbf{S}$ is an expression of the form $R(t_1, \ldots, t_n)$, where $R/n \in \mathbf{S}$, and $t_i \in \mathbf{C} \cup \mathbf{V}$ for each $i \in [n]$. Notice that a fact is an atom without variables. A *conjunctive query* (CQ) over $\mathbf{S}$ is a first-order formula of the form

$$Q(\bar{x}) \ := \ \exists \bar{y} \left(R_1(\bar{z}_1) \wedge \cdots \wedge R_n(\bar{z}_n) \right),$$

where each $R_i(\bar{z}_i)$, for $i \in [n]$, is an atom over $\mathbf{S}$, each variable mentioned in the $\bar{z}_i$'s appears either in $\bar{x}$ or $\bar{y}$, and $\bar{x}$ are the *answer variables* of Q. A *homomorphism* from a CQ Q as the one above to a database D is a function h from the set of variables and constants in Q to $\mathsf{dom}(D)$ that is the identity over $\mathbf{C}$ such that $R_i(h(\bar{z}_i)) \in D$ for every $i \in [n]$. A tuple $\bar{t} \in \mathsf{dom}(D)^{|\bar{x}|}$ is an *answer to Q over D* if there exists a homomorphism h from Q to D with $h(\bar{x}) = \bar{t}$. We write $Q(D)$ for the set of answers to Q over D. By abuse of notation, we may sometimes treat a CQ Q as the set of its atoms.

Repairs and Blocks. For an inconsistent database D w.r.t. a set Σ of primary keys, a *repair* of D w.r.t. Σ is a maximal subset of D that is consistent w.r.t. Σ, i.e., if $D' \subseteq D$ is a repair, then there is no $D'' \subseteq D$ that is consistent w.r.t. Σ and $D' \subsetneq D''$. Let $\mathsf{rep}(D, \Sigma)$ be the set of repairs of D w.r.t Σ. When we focus on primary keys, there is a convenient way to construct repairs. We first collect all the facts of the database that are in a conflict, i.e., they have the same predicate R and agree on $\mathsf{key}(R)$, into disjoint sets called blocks. A repair can be constructed by keeping exactly one fact from each block. Formally, for $\alpha = R(c_1, \ldots, c_n)$, the *key value* of α w.r.t. Σ is

$$\mathsf{key}_\Sigma(\alpha) \ = \begin{cases} \langle R, \langle c_1, \ldots, c_m \rangle \rangle & \text{if } \mathsf{key}(R) = \{1, \ldots, m\} \in \Sigma, \\ \\ \langle R, \langle c_1, \ldots, c_n \rangle \rangle & \text{otherwise.} \end{cases}$$

Then, given a database D, we define

$$\mathsf{block}_\Sigma(\alpha, D) \ = \ \{\beta \in D \mid \mathsf{key}_\Sigma(\beta) = \mathsf{key}_\Sigma(\alpha)\}.$$

Clearly, for $\alpha, \beta \in D$, if $\mathsf{key}_\Sigma(\alpha) = \mathsf{key}_\Sigma(\beta)$, then $\mathsf{block}_\Sigma(\alpha, D)$ and $\mathsf{block}_\Sigma(\beta, D)$ coincide. Moreover, if $\mathsf{block}_\Sigma(\alpha, D)$ is a singleton consisting of α, then α is not in a conflict. A repair can be constructed by keeping one fact from each block. Formally, with $\mathsf{block}_\Sigma(D)$ being the set of blocks $\{\mathsf{block}_\Sigma(\alpha, D) \mid \alpha \in D\}$,

$$\mathsf{rep}(D, \Sigma) = \left\{ \{\alpha_1, \ldots, \alpha_n\} \mid \langle \alpha_1, \ldots, \alpha_n \rangle \in \times_{B \in \mathsf{block}_\Sigma(D)} B \right\}.$$

Consistent Query Answering. We proceed to define the consistent answer to a CQ over a database that may be inconsistent w.r.t. a set of primary keys. This is based on the notion of relative frequency of a tuple. Consider a database D, a set Σ of primary keys, and a CQ $Q(\bar{x})$. The *relative frequency* of a tuple $\bar{t} \in \mathsf{dom}(D)^{|\bar{x}|}$ w.r.t. D, Σ and Q measures how often the tuple $\bar{t}$ is an answer to Q if we evaluate Q over all the repairs of D w.r.t. Σ. Formally, the relative frequency of $\bar{t}$ w.r.t. D, Σ and Q is the ratio

$$\mathsf{R}_{D,\Sigma,Q}(\bar{t}) \ = \ \frac{|\{D' \in \mathsf{rep}(D, \Sigma) \ \mid \ \bar{t} \in Q(D')\}|}{|\mathsf{rep}(D, \Sigma)|}.$$

Input: A database D, a set Σ of primary keys, a CQ $Q(\bar{x})$, $\epsilon > 0$, and $0 < \delta < 1$
Output: A set of tuple-number pairs

$ans := \emptyset$
foreach $\bar{t} \in \mathsf{dom}(D)^{|\bar{x}|}$ such that $\mathsf{R}_{D,\Sigma,Q}(\bar{t}) > 0$ **do**
  $p := \mathsf{ApxRelativeFreq}(D, \Sigma, Q, \bar{t}, \epsilon, \delta)$
  $ans := ans \cup \{(\bar{t}, p)\}$
return ans

Algorithm 1: ApxCQA[ApxRelativeFreq]

The numerator is simply the number of repairs $D' \in \mathsf{rep}(D, \Sigma)$ such that $\bar{t}$ is an answer to Q over D', while the denominator is the total number of repairs of D w.r.t. Σ. The *answer to Q over D in the presence of* Σ, denoted $\mathsf{ans}_{D,\Sigma}(Q)$, is defined as the set

$$\left\{ (\bar{t}, \mathsf{R}_{D,\Sigma,Q}(\bar{t})) \ \mid \ \bar{t} \in \mathsf{dom}(D)^{|\bar{x}|} \text{ and } \mathsf{R}_{D,\Sigma,Q}(\bar{t}) > 0 \right\}.$$

The main problem of concern in this context follows:

PROBLEM :	CQA
INPUT :	A database D, a set Σ of primary keys, and a CQ $Q(\bar{x})$.
OUTPUT :	The set $\mathsf{ans}_{D,\Sigma}(Q)$.

Computing the set $\mathsf{ans}_{D,\Sigma}(Q)$ boils down to iterating over each tuple $\bar{t} \in \mathsf{dom}(D)^{|\bar{x}|}$, and computing its relative frequency w.r.t. D, Σ and Q. This brings us to the following key problem:

PROBLEM :	RelativeFreq
INPUT :	A database D, a set Σ of primary keys, a CQ $Q(\bar{x})$, and a tuple $\bar{t} \in \mathsf{dom}(D)^{|\bar{x}|}$.
OUTPUT :	The number $\mathsf{R}_{D,\Sigma,Q}(\bar{t})$.

The above problem is intractable even when Σ and Q are fixed [6, 21]. Hence, also CQA is an intractable problem even when Σ and Q are fixed. To cope with the high complexity of CQA we give up exact query answering, and rely on data-efficient approximations.

3 APPROXIMATE CQA

An approximation of the relative frequency of a tuple is computed via a randomized approximation scheme for RelativeFreq. Having the latter in place, we can then talk about randomized approximation schemes for CQA. We assume familiarity with basic notions from probability theory such as the notions of event, random variable, and its expected value; details can be found in the appendix.

Approximation Schemes for RelativeFreq**.** A *data-efficient randomized approximation scheme* for RelativeFreq is a randomized algorithm A that takes a database D, a set Σ of primary keys, a CQ $Q(\bar{x})$, a tuple $\bar{t} \in \mathsf{dom}(D)^{|\bar{x}|}$, and numbers $\epsilon > 0$ and $0 < \delta < 1$, runs in polynomial time in $||D|| + ||\bar{t}||$, $1/\epsilon$ and $\log(1/\delta)$,[2] and produces a random variable $\mathsf{A}(D, \Sigma, Q, \bar{t}, \epsilon, \delta)$ such that

$$\Pr\left(|\mathsf{A}(D, \Sigma, Q, \bar{t}, \epsilon, \delta) - \mathsf{R}_{D,\Sigma,Q}(\bar{t})| \ \leq \ \epsilon \cdot \mathsf{R}_{D,\Sigma,Q}(\bar{t})\right) \ \geq \ 1 - \delta.$$

Approximation Schemes for CQA**.** A *data-efficient randomized approximation scheme* for CQA is an algorithm that takes a database D, a set Σ of primary keys, a CQ $Q(\bar{x})$, and numbers $\epsilon > 0$ and

[2] As usual, for a syntactic object o, we write $||o||$ for its size.

$0 < \delta < 1$, runs in polynomial time in $||D||$, $1/\epsilon$ and $\log(1/\delta)$, and computes a set of pairs of the form

$$\{(\bar{t}, \mathsf{A}(D, \Sigma, Q, \bar{t}, \epsilon, \delta)) \mid \bar{t} \in \mathsf{dom}(D)^{|\bar{x}|} \text{ and } \mathsf{R}_{D,\Sigma,Q}(\bar{t}) > 0\},$$

where A is a data-efficient randomized approximation scheme for RelativeFreq. It is easy to see that the next theorem holds for Algorithm 1, which is parametrized by an approximation scheme for the problem RelativeFreq. Note that checking if $\mathsf{R}_{D,\Sigma,Q}(\bar{t}) > 0$ can be done in polynomial time in $||D|| + ||\bar{t}||$: check whether there is a homomorphism h from $Q(\bar{x})$ to D such that $h(\bar{x}) = \bar{t}$ and $h(Q) \models \Sigma$.

Theorem 3.1. *Let* ApxRelativeFreq *be a data-efficient randomized approximation scheme for the problem* RelativeFreq. *It holds that* ApxCQA[ApxRelativeFreq] *is a data-efficient randomized approximation scheme for the problem* CQA.

Henceforth, we may simply say approximation scheme meaning data-efficient randomized approximation scheme.

4 APPROXIMATION SCHEMES

We present the main approximation schemes for RelativeFreq that can be obtained from the literature by adapting existing approximation schemes introduced in the context of DNF Boolean formulas. But first we need to introduce the crucial notion of synopsis.

4.1 Database Synopsis

In principle, an approximation scheme for RelativeFreq would require to access the input database D. However, doing this naively is prohibitive in practice since, in general, D is very large. It turns out that an approximation scheme for RelativeFreq does not really need to access the whole database, but only some relatively small parts of it, which we call synopsis, that depend on the set of primary keys, the CQ, and the given tuple.

Consider an instance of RelativeFreq, that is, a database D, a set Σ of primary keys, a CQ $Q(\bar{x})$, and a tuple $\bar{t} \in \mathsf{dom}(D)^{|\bar{x}|}$. The (Σ, Q)-*synopsis of* D *for* $\bar{t}$ is the pair $(\mathcal{H}, \mathcal{B})$, where

$$\mathcal{H} \;=\; \left\{ h(Q(\bar{x})) \;\middle|\; \begin{array}{l} h \text{ is a homomorphism from } Q(\bar{x}) \text{ to } D \\ \text{with } h(\bar{x}) = \bar{t}, \text{ and } h(Q) \models \Sigma \end{array} \right\}$$

and

$$\mathcal{B} \;=\; \{\mathsf{block}_\Sigma(\alpha, D) \mid \alpha \in \cup_{H \in \mathcal{H}} H\}\,.$$

In simple words, the (Σ, Q)-synopsis of D for $\bar{t}$ collects all the consistent homomorphic images of $Q(\bar{t})$ in D (the set $\mathcal{H}$), and the blocks of the atoms occurring in a consistent homomorphic image of $Q(\bar{t})$ in D (the set $\mathcal{B}$). We further define the set

$$\mathsf{db}(\mathcal{B}) \;=\; \{\{\alpha_1, \ldots, \alpha_n\} \;\mid\; \langle \alpha_1, \ldots, \alpha_n \rangle \in \times_{B \in \mathcal{B}} B\}\,,$$

i.e., the set of databases that can be formed by keeping exactly one atom from each member B of $\mathcal{B}$. We also define the ratio

$$\mathsf{R}_{(\mathcal{H},\mathcal{B})} \;=\; \frac{|\{I \in \mathsf{db}(\mathcal{B}) \;\mid\; H \subseteq I \text{ for some } H \in \mathcal{H}\}|}{|\mathsf{db}(\mathcal{B})|}$$

assuming that $\mathcal{H} \neq \emptyset$; otherwise, $\mathsf{R}_{(\mathcal{H},\mathcal{B})} = 0$.

We show a useful result, which essentially tells us that the (Σ, Q)-synopsis of a database D for a tuple $\bar{t}$ can be efficiently constructed (in the size of D and $\bar{t}$), and it contains enough information that allows us to compute the relative frequency of $\bar{t}$ w.r.t. D, Σ and Q.

Lemma 4.1. *Consider a database D, a set Σ of primary keys, a CQ $Q(\bar{x})$, and a tuple $\bar{t} \in \mathsf{dom}(D)^{|\bar{x}|}$. Let $(\mathcal{H}, \mathcal{B})$ be the (Σ, Q)-synopsis of D for $\bar{t}$. The following hold:*

(1) $(\mathcal{H}, \mathcal{B})$ can be computed in polynomial time in $||D|| + ||\bar{t}||$.
(2) For each $H \in \mathcal{H}$, $|H| \leq |Q|$.
(3) $\mathsf{R}_{D,\Sigma,Q}(\bar{t}) = \mathsf{R}_{(\mathcal{H},\mathcal{B})}$.
(4) $\mathsf{R}_{D,\Sigma,Q}(\bar{t}) = 0$ if and only if $\mathcal{H} = \emptyset$.

As we shall see, Lemma 4.1 allows an approximation scheme to directly operate on (Σ, Q)-synopses, thus significantly improving the performance. In order to be able to refer to (Σ, Q)-synopses, without explicitly mentioning the underlying database, key constraints, query, and tuple, we have the notion of admissible pairs. A pair of sets of databases $(\mathcal{H}, \mathcal{B})$ is called *admissible* if $\mathcal{H} \neq \emptyset$, and there are a database D, a set Σ of primary keys, a CQ $Q(\bar{x})$, and a tuple $\bar{t} \in \mathsf{dom}(D)^{|\bar{x}|}$ such that $(\mathcal{H}, \mathcal{B})$ is the (Σ, Q)-synopsis of D for $\bar{t}$; by definition, $\mathcal{B} \neq \emptyset$. For brevity, we may refer to admissible pairs $(\mathcal{H}, \mathcal{B})$ meaning that $\mathcal{H}$ and $\mathcal{B}$ are sets of databases.

4.2 Monte Carlo Approximation

Consider a randomized procedure Sample that takes as input an admissible pair $(\mathcal{H}, \mathcal{B})$, and computes a random number in $[0, 1]$. This randomized procedure actually samples objects from a sampling space obtained from $(\mathcal{H}, \mathcal{B})$, and checks whether each sampled object enjoys certain properties, which determines the output. Sample$((\mathcal{H}, \mathcal{B}))$ produces a random variable, and a crucial problem for us is computing its expected value $\mathbb{E}[\mathsf{Sample}((\mathcal{H}, \mathcal{B}))]$:

PROBLEM :	EV[Sample]
INPUT :	An admissible pair $(\mathcal{H}, \mathcal{B})$.
OUTPUT :	The number $\mathbb{E}[\mathsf{Sample}((\mathcal{H}, \mathcal{B}))]$.

We can talk about efficient randomized approximation schemes for EV[Sample] defined in a similar way as data-efficient randomized approximation schemes for RelativeFreq, but instead of the ratio $\mathsf{R}_{D,\Sigma,Q}(\bar{t})$ we consider the value $\mathbb{E}[\mathsf{Sample}((\mathcal{H}, \mathcal{B}))]$, while the randomized algorithm A is required to run in polynomial time in $|\mathcal{H}|$, $\max_{H \in \mathcal{H}}\{||H||\}$, $||\mathcal{B}||$, $1/\epsilon$ and $\log(1/\delta)$.[3] As we shall see, the following simple iterative procedure is an efficient randomized approximation scheme for EV[Sample] for a carefully chosen number of iterations N: (1) $S := 0$, (2) for N times do the following: $S := S +$ Sample$((\mathcal{H}, \mathcal{B}))$, and finally (3) return S/N.

It is clear that as long as we increase the number N of iterations in the above procedure, the ratio S/N is a better approximation of $\mathbb{E}[\mathsf{Sample}((\mathcal{H}, \mathcal{B}))]$. The crucial question is the following: what is a sufficiently large value for N? In particular, given $(\mathcal{H}, \mathcal{B})$, an error parameter ϵ, and an uncertainty coefficient δ, can we compute the *minimum* number (up to a constant factor) of iterations N such that the above algorithm approximates $\mathbb{E}[\mathsf{Sample}((\mathcal{H}, \mathcal{B}))]$ within a factor of ϵ, and with probability at least $1 - \delta$?

A positive answer to the above key question is obtained from [8], where a generic randomized algorithm has been proposed that optimally estimates, with high probability, such a number of iterations. Actually, from [8] we get a randomized algorithm, called OptEstimate[Sample], which is parameterized with a randomized

[3] It should run in polynomial time in $|\mathcal{H}|$, $\max_{H \in \mathcal{H}}\{||H||\}$ (not in $\max_{H \in \mathcal{H}}\{|H|\}$), and $||\mathcal{B}||$ since these are the parameters of a synopsis that depend on the database.

Input: An admissible pair $(\mathcal{H}, \mathcal{B})$, and $\epsilon > 0$ and $0 < \delta < 1$
Output: A random number in $[0, 1]$

$N :=$ OptEstimate[Sample]$((\mathcal{H}, \mathcal{B}), \epsilon, \delta)$
$S := 0$; $ctr := 0$
repeat
  $S := S +$ Sample$((\mathcal{H}, \mathcal{B}))$
  $ctr := ctr + 1$
until $ctr = N$;
return S/N

Algorithm 2: MonteCarlo[Sample]

procedure Sample as described above, that accepts as input $(\mathcal{H}, \mathcal{B})$, ϵ and δ, and optimally computes the number of iterations N. Having this algorithm in place, we then obtain the optimal Monte Carlo Estimator for $\mathbb{E}$[Sample$((\mathcal{H}, \mathcal{B}))$] shown in Algorithm 2. We know from [8] that if Sample enjoys certain properties, then we get that MonteCarlo[Sample] is an efficient approximation scheme for EV[Sample]; we write $||\mathcal{H}, \mathcal{B}||$ for $|\mathcal{H}| + \max_{H \in \mathcal{H}}\{||H||\} + ||\mathcal{B}||$:

Lemma 4.2. *Let* Sample *be a randomized procedure that takes as input an admissible pair $(\mathcal{H}, \mathcal{B})$, and outputs a number in* $[0, 1]$. *If (1)* Sample$((\mathcal{H}, \mathcal{B}))$ *runs in polynomial time in $||\mathcal{H}, \mathcal{B}||$, and (2) there exists a polynomial* pol *such that* $\mathbb{E}$[Sample$((\mathcal{H}, \mathcal{B}))] > 0$ *implies* $\mathbb{E}$[Sample$((\mathcal{H}, \mathcal{B}))] \geq 1/\text{pol}(||\mathcal{H}, \mathcal{B}||)$, *then the algorithm* MonteCarlo[Sample] *is an efficient randomized approximation scheme for the problem* EV[Sample].

Recall that our main concern are approximation schemes for RelativeFreq. Such approximation schemes can be obtained from the one for EV[Sample] discussed above. This is achieved by devising a randomized procedure Sample that satisfies the items (1) and (2) in Lemma 4.2, while the value $\mathsf{R}_{(\mathcal{H}, \mathcal{B})}$, for an admissible pair $(\mathcal{H}, \mathcal{B})$, coincides with $\mathbb{E}$[Sample$((\mathcal{H}, \mathcal{B}))$], or it can be derived from $\mathbb{E}$[Sample$((\mathcal{H}, \mathcal{B}))$]. More precisely, the goal is to devise a randomized procedure Sample, which takes as input an admissible pair $(\mathcal{H}, \mathcal{B})$ and outputs a number in $[0, 1]$, that is *r-good* for some rational number $r > 0$, i.e., it satisfies conditions (1) and (2) in Lemma 4.2, r can be computed in polynomial time in $||\mathcal{H}, \mathcal{B}||$, and $\mathbb{E}$[Sample$((\mathcal{H}, \mathcal{B}))] = \mathsf{R}_{(\mathcal{H}, \mathcal{B})} \cdot r$. Having such a procedure in place, it is then not difficult to obtain an approximation scheme for RelativeFreq due to Lemma 4.1. We proceed to present three such randomized procedures, which in turn give rise to three Monte Carlo approximation schemes for RelativeFreq. We call those procedures *samplers*. The first one samples from the natural sampling space, while the other two from a symbolic sampling space.

Natural Sampling Space. For an admissible pair $(\mathcal{H}, \mathcal{B})$, the natural sampling space $\mathcal{S}_{(\mathcal{H}, \mathcal{B})}$ is defined as the set of databases db$(\mathcal{B})$. Our first sampler, dubbed SampleNatural, is given in Sampler 1.

Lemma 4.3. SampleNatural$((\mathcal{H}, \mathcal{B}))$ *is* 1*-good.*

From Lemmas 4.1, 4.2, 4.3, we get the following for Algorithm 3:

Theorem 4.4. Natural *is a data-efficient randomized approximation scheme for* RelativeFreq.

Symbolic Sampling Space: The KL Variation. An alternative approach towards an approximation scheme for RelativeFreq is to use a sampler such that, on input $(\mathcal{H}, \mathcal{B})$, its expected value is not exactly $\mathsf{R}_{(\mathcal{H}, \mathcal{B})}$, but a number from which $\mathsf{R}_{(\mathcal{H}, \mathcal{B})}$ can be derived. To this end, we devise a slightly more complex sampling space than the natural one called symbolic (a term borrowed from [24]), by exploiting ideas from [15]. Consider an admissible pair $(\mathcal{H}, \mathcal{B})$. Let $H_1, \ldots, H_n$ be an arbitrary ordering (e.g., lexicographical) among the databases of $\mathcal{H}$. For each $i \in [n]$, we define the set

$$\mathcal{I}^i_{(\mathcal{H}, \mathcal{B})} = \{I \in \text{db}(\mathcal{B}) \mid H_i \subseteq I\}.$$

Intuitively, if $(\mathcal{H}, \mathcal{B})$ is the (Σ, Q)-synopsis of D for $\bar{t}$, then $\mathcal{I}^i_{(\mathcal{H}, \mathcal{B})}$ collects all the databases $I \in \text{db}(\mathcal{B})$ for which H_i is a witness of $\bar{t} \in Q(I)$. Our symbolic sampling space is defined as

$$\mathcal{S}^\bullet_{(\mathcal{H}, \mathcal{B})} = \left\{(i, I) \mid i \in [n] \text{ and } I \in \mathcal{I}^i_{(\mathcal{H}, \mathcal{B})}\right\}.$$

Our second sampler, called SampleKL, is given in Sampler 2. Then:

Lemma 4.5. SampleKL$((\mathcal{H}, \mathcal{B}))$ *is* $\left(|\text{db}(\mathcal{B})|/|\mathcal{S}^\bullet_{(\mathcal{H}, \mathcal{B})}|\right)$*-good.*

From Lemmas 4.1, 4.2 and 4.5, we get the following result for the approximation scheme KL depicted in Algorithm 4:

Theorem 4.6. KL *is a data-efficient randomized approximation scheme for* RelativeFreq.

Input: An admissible pair $(\mathcal{H}, \mathcal{B})$
Output: A random number in $[0, 1]$

Choose $I \in \mathcal{S}_{(\mathcal{H}, \mathcal{B})}$ with probability $\dfrac{1}{|\mathcal{S}_{(\mathcal{H}, \mathcal{B})}|}$
if there exists $H \in \mathcal{H}$ such that $H \subseteq I$ **then**
  return 1
else
  return 0

Sampler 1: SampleNatural

Input: A database D, a set Σ of primary keys, a CQ $Q(\bar{x})$, a tuple $\bar{t} \in \text{dom}(D)^{|\bar{x}|}$, $\epsilon > 0$, and $0 < \delta < 1$
Output: A random number in $[0, 1]$

Let $(\mathcal{H}, \mathcal{B})$ be the (Σ, Q)-synopsis of D for $\bar{t}$
if $\mathcal{H} \neq \emptyset$ **then**
  $p :=$ MonteCarlo[SampleNatural]$((\mathcal{H}, \mathcal{B}), \epsilon, \delta)$
else
  $p := 0$
return p

Algorithm (Approximation Scheme) 3: Natural

Input: An admissible pair $(\mathcal{H}, \mathcal{B})$
Output: A random number in $[0, 1]$

Choose $(i, I) \in \mathcal{S}^\bullet_{(\mathcal{H}, \mathcal{B})}$ with probability $\dfrac{1}{|\mathcal{S}^\bullet_{(\mathcal{H}, \mathcal{B})}|}$
if there is no $j < i$ such that $I \in \mathcal{I}^j_{(\mathcal{H}, \mathcal{B})}$ **then**
  return 1
else
  return 0

Sampler 2: SampleKL

Input: A database D, a set Σ of primary keys, a CQ $Q(\bar{x})$, a tuple $\bar{t} \in \mathsf{dom}(D)^{|\bar{x}|}$, $\epsilon > 0$, and $0 < \delta < 1$
Output: A random number in $[0, 1]$

Let $(\mathcal{H}, \mathcal{B})$ be the (Σ, Q)-synopsis of D for $\bar{t}$
if $\mathcal{H} \neq \emptyset$ **then**
  $p := \mathsf{MonteCarlo}[\mathsf{sampleKL(M)}]((\mathcal{H}, \mathcal{B}), \epsilon, \delta) \cdot \frac{|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|}{|\mathsf{db}(\mathcal{B})|}$
else
  $p := 0$
return p.

Algorithm (Approximation Scheme) 4: KL(M)

Input: An admissible pair $(\mathcal{H}, \mathcal{B})$
Output: A random number in $[0, 1]$

Choose $(i, I) \in \mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}$ with probability $\frac{1}{|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|}$
$k := 0$
foreach $j \in \{1, \ldots, |\mathcal{H}|\}$ **do**
  if $I \in \mathcal{I}^{j}_{(\mathcal{H},\mathcal{B})}$ **then**
    $k := k + 1$
return $\frac{1}{k}$

Sampler 3: SampleKLM

Symbolic Sampling Space: The KLM Variation. Our third sampler is a variation of SampleKL, called SampleKLM, inspired by an algorithm presented in [26], and is given in Sampler 3. As expected, Lemma 4.5 holds also for this variation:

Lemma 4.7. SampleKLM$((\mathcal{H}, \mathcal{B}))$ *is* $\left(|\mathsf{db}(\mathcal{B})|/|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|\right)$*-good.*

From Lemmas 4.1, 4.2, 4.7, we get the following result for the approximation scheme KLM depicted in Algorithm 4:

Theorem 4.8. KLM *is a data-efficient randomized approximation scheme for* RelativeFreq.

Let us stress that, although SampleKL and SampleKLM are both $|\mathsf{db}(\mathcal{B})|/|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|$-good, there are crucial differences between the two that might make one preferable over the other in some contexts. In general, SampleKLM can be much slower than SampleKL since to generate a sample it *always* needs to iterate over every element of $\mathcal{H}$, for a given admissible pair $(\mathcal{H}, \mathcal{B})$, whereas the number of iterations in SampleKL, which depends on $(\mathcal{H}, \mathcal{B})$, is generally smaller that $|\mathcal{H}|$. However, the variance of the random variable SampleKLM$((\mathcal{H}, \mathcal{B}))$ is generally smaller than the variance of SampleKL$((\mathcal{H}, \mathcal{B}))$, which can effectively reduce the time required to optimally estimate the number of samples needed by means of the algorithm OptEstimate[SampleKLM].

4.3 Self-adjusting Approximation

In this section, we follow a different approach for devising an approximation scheme for RelativeFreq, inspired by results from [15]. A key difference of this approach is that the right number of iterations is computed deterministically, i.e., via a deterministic instead of a randomized procedure. Although this estimation is not optimal as the one discussed in the previous section for Monte Carlo approximations schemes, it has the advantage that it makes the running time of the relative approximation scheme more predictable.

Input: A database D, a set Σ of primary keys, a CQ $Q(\bar{x})$, a tuple $\bar{t} \in \mathsf{dom}(D)^{|\bar{x}|}$, $\epsilon > 0$, and $0 < \delta < 1$
Output: A random number in $[0, 1]$

Let $(\mathcal{H}, \mathcal{B})$ be the (Σ, Q)-synopsis of D for $\bar{t}$
if $\mathcal{H} \neq \emptyset$ **then**
  $p := \mathsf{SelfAdjustingCoverage}((\mathcal{H}, \mathcal{B}), \epsilon, \delta) \cdot \frac{1}{|\mathsf{db}(\mathcal{B})|}$
else
  $p := 0$
return p

Algorithm (Approximation Scheme) 5: Cover

The algorithm that we are going to use is known in the literature as the *self-adjusting coverage algorithm*, which has been originally proposed for solving the *union of sets problem* [15]. The latter takes as input the description of $n \geq 1$ sets $S_1, \ldots, S_n$, and asks for the number $|\bigcup_{i \in [n]} S_i|$. The following is a problem of special interest for us, which is essentially the union of sets problem tailored to our setting. Recall that, given an admissible pair $(\mathcal{H}, \mathcal{B})$, we assume an arbitrary ordering $H_1, \ldots, H_n$ among the databases of $\mathcal{H}$.

PROBLEM :	UnionOfSets
INPUT :	An admissible pair $(\mathcal{H}, \mathcal{B})$.
OUTPUT :	The number $\lvert \bigcup_{i \in [n]} \mathcal{I}^{i}_{(\mathcal{H},\mathcal{B})} \rvert$.

Observe that the above problem asks for the numerator of the ratio $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$. Hence, by having an efficient randomized approximation scheme for UnionOfSets, we get an approximation scheme for RelativeFreq due to Lemma 4.1. An efficient approximation scheme for UnionOfSets, dubbed SelfAdjustingCoverage, can be obtained via the self-adjusting coverage algorithm in [15], and can be found in the appendix. The next result concerning Algorithm 5 follows:

Theorem 4.9. Cover *is a data-efficient randomized approximation scheme for* RelativeFreq.

5 IMPLEMENTATION DETAILS

We now discuss our implementation of the approximation schemes for CQA. It is crucial to clarify that the implementation slightly deviates from the pseudocode given in Algorithm 1 for efficiency reasons. In particular, each call to ApxRelativeFreq$(D, \Sigma, Q, \bar{t}, \epsilon, \delta)$ in Algorithm 1 computes the (Σ, Q)-synopsis of D for $\bar{t}$ separately. This is prohibitive since computing a synopsis needs to query the underlying database, and thus, access the disk several times. Our implementation avoids redundant disk accesses by computing all the (Σ, Q)-synopses via a single SQL query. Given a database D, a set Σ of primary keys, a CQ $Q(\bar{x})$, and $\epsilon > 0$ and $0 < \delta < 1$, in our implementation we first construct, via a preprocessing step,

$$\mathsf{syn}_{\Sigma,Q}(D) = \left\{ (\bar{t}, (\mathcal{H}, \mathcal{B})) \;\middle|\; \begin{array}{l} \mathsf{R}_{D,\Sigma,Q}(\bar{t}) > 0, \text{and } (\mathcal{H}, \mathcal{B}) \text{ is} \\ \text{the } (\Sigma, Q)\text{-synopsis of } D \text{ for } \bar{t} \end{array} \right\}.$$

We then iterate over the pairs $(\bar{t}, (\mathcal{H}, \mathcal{B}))$ from $\mathsf{syn}_{\Sigma,Q}(D)$ and call the approximation scheme ApxRelativeFreq. Notice, however, that

ApxRelativeFreq does not have to explicitly compute the (Σ, Q)-synopsis of D for $\bar{t}$, and then check whether the set of homomorphic images is non-empty, since this has been done by the preprocessing step.[4] ApxRelativeFreq only has to compute the number p. The interesting task here was the implementation of the preprocessing step that needs to effectively compute the set $\mathsf{syn}_{\Sigma,Q}(D)$, while the rest was implemented in a rather straightforward way.

The Preprocessing Step. A simple but useful observation is that all the approximation schemes presented in Section 4 that take as input an admissible pair $(\mathcal{H}, \mathcal{B})$, which is essentially the (Σ, Q)-synopsis of D for some tuple $\bar{t}$, are oblivious to the syntactic shape of the facts in $\mathcal{H}$ and $\mathcal{B}$, i.e., the actual relation and tuple of constants of those facts is irrelevant to the execution of the algorithms. This allows us to work with an encoding of $\mathsf{syn}_{\Sigma,Q}(D)$, denoted $\mathsf{enc}(\mathsf{syn}_{\Sigma,Q}(D))$, using integer identifiers for facts. It turned out that such an encoding can be constructed by first executing a simple rewriting Q^{rew} of Q over D, and then obtain $\mathsf{enc}(\mathsf{syn}_{\Sigma,Q}(D))$ from $Q^{\mathsf{rew}}(D)$ in linear time in the size of $Q^{\mathsf{rew}}(D)$. Due to space constraints, we omit the detailed definition of Q^{rew} as an SQL query, as well as how the set $\mathsf{enc}(\mathsf{syn}_{\Sigma,Q}(D))$ is obtained from $Q^{\mathsf{rew}}(D)$. The details can be found in the appendix.

Implementing the Rest. For implementing the rest of the approximation schemes in question, we extended the framework *Approximate DNF Counting Suite* (ADCS) from [24], which collects algorithms for approximating the number of satisfying assignments of DNF Boolean formulas. This suite provides implementations of Algorithm 2 and the self-adjusting coverage algorithm. Hence, it was enough to implement Samplers 1, 2 and 3. The approximation scheme for CQA (Algorithm 1) was implemented as a family of algorithms (one for each approximation scheme for RelativeFreq), where the set $\mathsf{enc}(\mathsf{syn}_{\Sigma,Q}(D))$ is constructed in a preprocessing step as above. For making random choices, we used the Mersenne Twister pseudo-random number generator from [23].

6 EXPERIMENTAL SETTING

In order to effectively evaluate the approximation schemes for CQA presented above, we need to design our test scenarios in a way that will allow us to expose how the running time of the approximation schemes is affected by key input parameters such as the amount of inconsistency in the database, and the number of joins in the query. In Section 6.1, we discuss the tools that we used to generate our test scenarios. In Section 6.2, we describe our test scenarios. Finally, in Section 6.3, we give details about the experimental setup.

6.1 Test Scenarios Infrastructure

For devising our test scenarios, we need a way to generate data, add some inconsistency (we also call it noise) to the data, and generate CQs. For generating the data, we solely rely on existing tools. For adding noise to the data, although several different tools exist that could help us to accomplish this task, none of those tools provides the level of control that is needed for our analysis, and thus, we devised our own noise generator. For generating CQs, we use an existing tool that allows us to tune syntactic (also called static) parameters of the query (e.g., the number of joins), together with our generator that allows us to tune also some dynamic (database dependent) parameters that are central for our analysis.

Data Generator. For the data generation phase we exploit TPC-H (2.18.0), one of the main TPC benchmarks that offers a diverse workload of manually curated queries over a relational schema, modelling most common data management scenarios. The schema is in third normal form, and models a typical data warehouse, dealing with sales, customers, and suppliers. It contains eight relations, and comes with integrity constraints, including primary keys. Databases of varied size can be generated via the data generator provided by TPC-H. It receives a *scale factor* as part of its input, specifying the size of the database to generate, and it generates a NULL-free database. Note, however, that the databases generated with this tool are consistent w.r.t. the underlying constraints. Nevertheless, since it has been designed to generate data that is as close as possible to a real-world scenario, we use it for generating consistent data, and then inject inconsistency via a noise generator that we developed.

Query-aware Noise Generator for Primary Keys. Making a database inconsistent is a complex process, and several attempts for developing a general-purpose tool for this task can be found in the literature; see, e.g., [2, 4, 5, 9, 12]. Nevertheless, for the reasons explained below, none of those tools is suitable for our purposes:

► Existing tools are query-oblivious, i.e., do not take into account any query. The reason is that they have been developed in the context of data cleaning where no query is involved. However, by generating noise in a query-oblivious way, we may fail to obtain meaningful datasets for our purposes. Even if we generate noise by randomly adding facts to the database, considering only the primary keys, it is likely that we will not affect the evaluation of the query. This is because we typically deal with very large databases, while only a small portion of them is needed to answer a query.

► Existing tools have been designed to work with more general classes of constraints than primary keys such as equality-generating dependencies. Hence, although they allow us to tune some parameters during the noise generation process, we cannot tune fine-grained parameters that are specific to primary keys such as the number of facts in a block, or the number of non-singleton blocks.

Our noise generator takes as input a database D, a set Σ of primary keys with $D \models \Sigma$, a CQ $Q(\bar{x})$ such that $Q(D) \neq \emptyset$, a number $0 < p \leq 1$, and two integers ℓ, u with $0 < \ell \leq u$. The number p specifies the percentage of noise (w.r.t. Σ) that should be added to D, which will lead to a database D^*, while ℓ and u specify the minimum and the maximum size of a non-singleton block in $\mathsf{block}_\Sigma(D^*)$. The following steps are performed:

Step 1: The noise generator first constructs the set

$$\mathsf{syn}_{\Sigma,Q}(D) = \{(\bar{t}_1, (\mathcal{H}_1, \mathcal{B}_1)), \ldots, (\bar{t}_n, (\mathcal{H}_n, \mathcal{B}_n))\}$$

as in Section 5. Recall that, for $i \in [n]$, $(\bar{t}_i, (\mathcal{H}_i, \mathcal{B}_i))$ is such that $\bar{t}_i \in Q(D)$, $\mathcal{H}_i$ is the set of homomorphic images of $Q(\bar{t}_i)$ in D, and $\mathcal{B}_i$ is the set of blocks of the atoms occurring in a homomorphic image of $Q(\bar{t}_i)$ in D. Thus, $\mathsf{syn}_{\Sigma,Q}(D)$, and in particular $\mathcal{H} = \bigcup_{i=1}^{n} \mathcal{H}_i$, contains all the facts of D that can affect the query result.

Step 2: For every relation R in $\mathcal{H}$, with $\mathsf{key}(R) = \{1, \ldots, k\} \in \Sigma$, assuming that $\mathcal{H}_R$ is the set of R-facts in $\mathcal{H}$ – there are no two R-facts in $\mathcal{H}_R$ with the same key value – the noise generator randomly

[4] By Lemma 4.1, for each $(\bar{t}, (\mathcal{H}, \mathcal{B})) \in \mathsf{syn}_{\Sigma,Q}(D)$, $\mathcal{H} \neq \emptyset$.

selects $\lceil p \cdot |\mathcal{H}_R| \rceil$ facts from $\mathcal{H}_R$. Let $R(\bar{a}_1, \bar{b}_1), \ldots, R(\bar{a}_m, \bar{b}_m)$, for $m > 0$, be the selected facts and $\langle R, \bar{a}_1 \rangle, \ldots, \langle R, \bar{a}_m \rangle$ the key values.

Step 3: Finally, for each $i \in [m]$, the noise generator chooses a number $s \in [\ell, u]$, uniformly at random, which essentially specifies the size of the block of facts with key value $\langle R, \bar{a}_i \rangle$, and adds $s - 1$ new facts $R(\bar{a}_i, \bar{u}_1), \ldots, R(\bar{a}_i, \bar{u}_{s-1})$ to the database D.

It remains to explain how the new facts added to D at step 3 are generated. One could naively construct the new facts by randomly constructing each tuple of constants $\bar{u}_j$, for $j \in [s-1]$. However, tuples with randomly generated values are unlikely to join with other tuples in the database. Thus, most probably, such randomly generated facts will not be part of the (Σ, Q)-synopsis of the new inconsistent database D^* for some arbitrary tuple, which means that they will not affect the query result over D^*. Hence, our noise generator constructs the above $s - 1$ facts in a more careful way. For each $j \in [s-1]$, it randomly selects an atom $R(\bar{a}', \bar{u}') \in D$ such that the key value of $R(\bar{a}', \bar{u}')$, that is, $\langle R, \bar{a}' \rangle$, is different from the key value of the block under construction, say $\langle R, \bar{a}_i \rangle$ for $i \in [m]$, and then it sets $\bar{u}_j = \bar{u}'$. Selecting values in this way ensures that the noise generator preserves the join patterns present in the data. This is especially true for joins over multi-attribute foreign-keys.

Query Generator. The TPC-H benchmark provides several manually curated queries. However, we need a range of carefully designed queries that will allow us to stress the approximation schemes.

Static Query Generator. To generate our stress test queries we exploit a recent query generator [3], which we call *static query generator* (SQG) since it allows us to tune only static parameters of the query, without taking into account any database. In particular, SQG takes as input a schema $\mathbf{S}$ (the schema of the output CQ), two integers $j \geq 0$ and $c \geq 0$ (the number of joins, and the number of occurrences of constant values, respectively, in the output CQ), a number $0 \leq p \leq 1$ (the percentage of attributes in the output CQ that should be projected), and a function f from $\{R[k] \mid R/n \in \mathbf{S} \text{ and } k \in [n]\}$ to $\mathbf{C}$, i.e., from the set of attributes of the relations of $\mathbf{S}$ ($R[k]$ refers to the k-th attribute of R) to $\mathbf{C}$, which specifies the constant values that can appear in a certain attribute. Details on how the output CQ is generated can be found in the appendix.

Dynamic Query Generator. One of the challenges of our work is to identify the right input parameters that allow us to properly analyze the behaviour of the approximation schemes for CQA in question. Concerning the input query, the main parameters that one can consider are, as customary in the literature, the number of joins, the number of occurrences of constants, and the number of attributes that are projected; these are the parameters that SQG allows us to tune. However, by considering only those static parameters we cannot control the size of a database synopsis, which clearly affects the performance of the approximation schemes. It turned out that we also need to consider some dynamic query parameters.

Consider a database D, a set Σ of primary keys, and a CQ $Q(\bar{x})$. Assume that $\mathsf{syn}_{\Sigma,Q}(D) = \{(\bar{t}_i, (\mathcal{H}_i, \mathcal{B}_i))\}_{i \in [n]}$ for some $n \geq 1$; as usual, $(\bar{t}_i, (\mathcal{H}_i, \mathcal{B}_i))$ is such that $\bar{t}_i \in Q(D)$, $\mathcal{H}_i$ is the set of homomorphic images of $Q(\bar{t})$ in D, and $\mathcal{B}_i$ is the set of blocks of the atoms occurring in a homomorphic image of $Q(\bar{t})$ in D. The dynamic query parameters that are central for our analysis are: (i) the *homomorphic size* of Q w.r.t. D defined as $|\bigcup_{i=1}^{n} \mathcal{H}_i|$, which measures how large is the portion of D that is needed to compute $Q(D) = \{\bar{t}_1, \ldots, \bar{t}_n\}$, and (ii) the *output size* of Q w.r.t. D defined as the cardinality of $Q(D)$, which coincides with $|\mathsf{syn}_{\Sigma,Q}(D)|$.

It is clear that the output size of Q w.r.t. D affects all the approximation schemes in exactly the same way; in fact, it only affects the number of iterations over the tuples of $Q(D)$ (see Algorithm 1). Hence, this parameter alone will not provide useful insights in understanding the key differences between the various approximation schemes. The homomorphic size of Q w.r.t. D, on the other hand, is more interesting as it affects the size of a (Σ, Q)-synopsis $(\mathcal{H}, \mathcal{B})$ in $\mathsf{syn}_{\Sigma,Q}(D)$. In particular, it will affect the running time of the approximation schemes for RelativeFreq that, according to our implementation of Algorithm 1, we need to call with input $(\mathcal{H}, \mathcal{B})$ at each iteration. Hence, since each (Σ, Q)-synopsis in $\mathsf{syn}_{\Sigma,Q}(D)$ can have different size, it is natural to consider the *average* size of a (Σ, Q)-synopsis in $\mathsf{syn}_{\Sigma,Q}(D)$ as a key parameter, which is given by the formula $|\bigcup_{i=1}^{n} \mathcal{H}_i| / |\mathsf{syn}_{\Sigma,Q}(D)|$, i.e., the average number of homomorphic images of $\bigcup_{i=1}^{n} \mathcal{H}_i$ that are distributed to each (Σ, Q)-synopsis of $\mathsf{syn}_{\Sigma,Q}(D)$. Let us recall, however, that one of the main goals of our experimental analysis is to provide hints on which approximation scheme is the indicated one based on some key input parameters, which now include the average size of a (Σ, Q)-synopsis. Nevertheless, we would not be able to provide such a general guidance, based on the absolute value of this average, as it can be arbitrarily large depending on D, Σ, and Q. Hence, we consider a parameter, which we call the *balance of Q w.r.t. D*, defined as the inverse of the average above. This means that the closer is the balance to 1, the smaller is the average size of a (Σ, Q)-synopsis, and the closer it is to 0, the higher is this average.

It should be clear that we need a query generator that allows us to tune, not only static query parameters, but also dynamic ones, and in particular the balance. We describe such a tool, called *dynamic query generator* (DQG). It takes as input an integer $n > 0$, a database D, a CQ Q (the starting query from which n new CQs should be generated with varied balance w.r.t. D), numbers $0 \leq b_1, \ldots, b_n \leq 1$ (the balance of the output queries w.r.t. D), and an integer $t > 0$ (a time threshold that indicates how many hours the query generator should run). The goal is to generate CQs $Q_1, \ldots, Q_n$ starting from Q such that, for each $i \in [n]$, the balance of Q_i w.r.t. D is as close as it can get to b_i in the available time t. In particular, DQG generates a pool of CQs P by iteratively choosing at random a subset of the attributes of the relations occurring in Q that are projected, and this is repeated for t hours. Then, for each $i \in [n]$, the generator keeps a CQ $Q_i \in P$ such that, for each $Q' \in P$, the absolute difference between the balance of Q_i w.r.t. D and b_i is less or equal than the absolute difference between the balance of Q' w.r.t. D and b_i, i.e., the balance of Q_i is closer to b_i than the balance of the other CQs.

6.2 Test Scenarios

For our test scenarios we consider the TPC-H schema, denoted $\mathbf{S}_{\mathrm{H}}$, and the set Σ_{H} of primary keys over $\mathbf{S}_{\mathrm{H}}$ coming with the TPC-H benchmark. A test scenario is a family of pairs of the form (D, Q), where D is a database over $\mathbf{S}_{\mathrm{H}}$ that is inconsistent w.r.t. Σ_{H}, and Q is a CQ over $\mathbf{S}_{\mathrm{H}}$. The database-query pairs that form a test scenario T are carefully chosen depending on which aspect of the approximation schemes for CQA we want to investigate via T. For example,

a test scenario that is meant to analyze how the approximation schemes behave when the amount of noise is varied, should consist of database-query pairs where the amount of noise in the databases varies, while the main query parameters such as the number of joins in a CQ, and the balance of a CQ w.r.t. its paired database, should be fixed. We first generated a large set of database-query pairs and then defined our test scenarios as subsets of this large set.

A Large Set of Database-Query Pairs. Starting from $\mathbf{S}_\mathrm{H}$ and Σ_H, we generated a large set of database-query pairs P_H as follows:

(1) First, we generated a database D_H over $\mathbf{S}_\mathrm{H}$, with $D_\mathrm{H} \models \Sigma_\mathrm{H}$, by exploiting the data generation tool provided by the TPC-H benchmark with scale factor 1GB; D_H contains almost 9 millions tuples.

(2) We then generated 25 CQs using SQG. In fact, for each $j \in [5]$, we generated five CQs $Q_j^1, \ldots, Q_j^5$ by iteratively calling SQG with input $(\mathbf{S}_\mathrm{H}, j, 2, 1, f)$, where f maps an attribute $R[i]$, for $R/n \in \mathbf{S}_\mathrm{H}$ and $i \in [n]$, to the set of constants occurring in D_H at attribute $R[i]$, and keeping the CQs whose evaluation over D_H is non-empty. In other words, for each $j \in [5]$, we generated five CQs over $\mathbf{S}_\mathrm{H}$ with j joins, 2 occurrences of constants, and all the attributes being projected, that are non-empty over D_H. After several experiments, we observed that almost all the CQs generated by SQG with less than 2 or greater than 2 occurrences of constants are trivial, i.e, when evaluated over D_H, in the former case they return everything that can be returned, while in the latter case are empty. This is why we focus on CQs with 2 occurrences of constants.

(3) For each $Q \in \{Q_j^i\}_{i,j\in[5]}$, we generated the databases

$$D_Q[0.1], D_Q[0.2], \ldots, D_Q[0.9], D_Q[1],$$

which are inconsistent w.r.t. Σ_H, by calling, for each number $p \in \{0.1, \ldots, 0.9, 1\}$, the query-aware noise generator with the input $(D_\mathrm{H}, \Sigma_\mathrm{H}, Q, p, 2, 5)$. Recall that the numbers 2, 5 specify the min. and max. size of a non-singleton block in $\mathrm{block}_\Sigma(D_Q[p])$. Sizes outside the range $[2, 5]$ did not influence the results of our analysis.

(4) For each CQ $Q \in \{Q_j^i\}_{i,j\in[5]}$ and $p \in \{0.1, 0.2, \ldots, 0.9, 1\}$, we generated 11 CQs

$$Q_p[0], Q_p[0.1], Q_p[0.2], \ldots, Q_p[0.9], Q_p[1],$$

where $Q_p[0]$ is the Boolean query obtained from Q by having all the variables as existentially quantified, while the other 10 queries are obtained by DQG with input $(10, D_Q[p], 0.1, 0.2, \ldots, 0.9, 1, 12)$. In other words, the CQ $Q_p[q]$, for $q \in \{0.1, 0.2, \ldots, 0.9, 1\}$, is a query such that its balance w.r.t. $D_Q[p]$ is close to q (in fact, as close as it could get after running DQG for 12 hours).

Having the databases from step 3 and the queries from step 4, P_H is defined as the set of database-query pairs

$$\big\{(D_Q[p], Q_p[q]) \mid Q \in \{Q_j^i\}_{i,j\in[5]}, \\ p \in \{0.1, \ldots, 0.9, 1\} \text{ and } q \in \{0, 0.1, \ldots, 0.9, 1\}\big\},$$

which consists of 2750 database-query pairs.

The Various Test Scenarios. Having P_H in place, we can now introduce the test scenarios that we considered in our experiments. It should be clear by now that the input parameters analyzed in our evaluation are (i) the amount of noise in the input inconsistent database (w.r.t. Σ_H), (ii) the balance of the input query (w.r.t. to the input database), and (iii) the number of joins in the input query. In order to understand how the running time of the approximation schemes for CQA is affected by the above parameters, we considered different test scenarios (which are subsets of P_H) where one of the parameters varies and the other two are kept fixed:

► For each balance value $q \in \{0, 0.1, \ldots, 0.9, 1\}$ and number of joins $j \in [5]$, we considered the *noise scenario* with 50 pairs

$$\mathsf{Noise}[q, j] = \bigcup_{\substack{p\in\{0.1,\ldots,0.9,1\} \\ Q\in\{Q_j^i\}_{i\in[5]}}} \big\{(D_Q[p], Q_p[q])\big\}.$$

► For each noise percentage $p \in \{0.1, \ldots, 0.9, 1\}$ and number of joins $j \in [5]$, we considered the *balance scenario* with 55 pairs

$$\mathsf{Balance}[p, j] = \bigcup_{\substack{q\in\{0,0.1,\ldots,0.9,1\} \\ Q\in\{Q_j^i\}_{i\in[5]}}} \big\{(D_Q[p], Q_p[q])\big\}.$$

► For each noise percentage $p \in \{0.1, \ldots, 1\}$ and balance value $q \in \{0, 0.1, \ldots, 1\}$, we considered the *joins scenario* with 25 pairs

$$\mathsf{Joins}[p, q] = \bigcup_{Q\in\{Q_j^i\}_{i,j\in[5]}} \big\{(D_Q[p], Q_p[q])\big\}.$$

Summing up, we considered 55 noise scenarios each consisting of 50 database-query pairs, 50 balance scenarios each consisting of 55 pairs, and 110 join scenarios each consisting of 25 pairs.

6.3 Experimental Setup

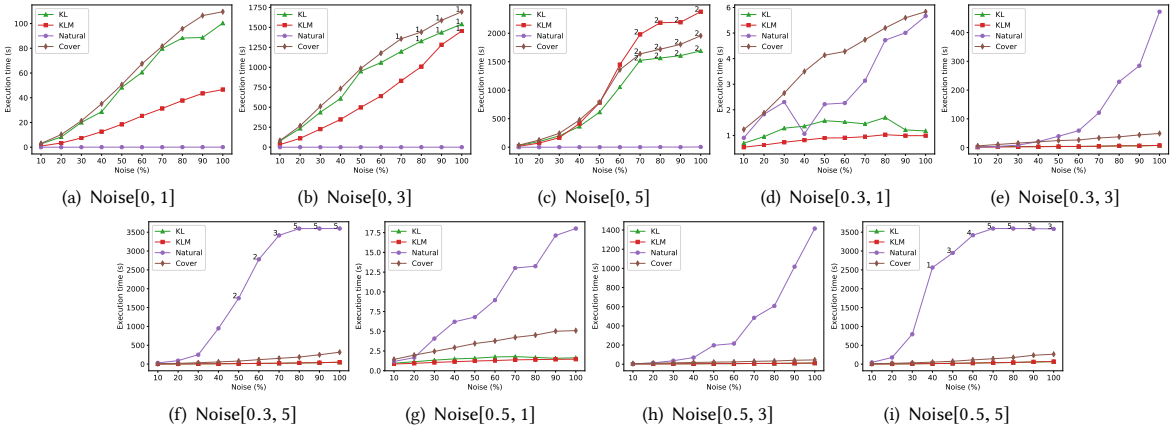
Hardware and Software Configuration. For the experiments we used two HP EliteDesk 800 G4 SFF Desktops, with an Intel(R) Core(TM) i5-8500 CPU@3.00GHz processor, 16GB RAM, 500GB mechanical drive, running Xubuntu 19.04 64-bit. All the databases of our test scenarios are stored in each machine in a single PostgreSQL 11.5 instance. All the algorithms (i.e, the approximation schemes, including the preprocessing step) have been implemented in C++, compiled with g++ 8.3.0 using the -std=c++11 flag and the -O3 optimization flag; no other flags have been used.

Execution. The experiments have been performed with $\delta = 0.25$ and $\epsilon = 0.1$, i.e., 75% confidence and 10% error. The reason why we fix the value of ϵ and δ, which are problem-agnostic parameters, is because we know that their actual value does not allow us to reliably differentiate the approximation schemes [24]. All the approximation schemes were required, for each test scenario, to terminate within 1 hour; beyond that, they were flagged as timed out.

CPU Time and Data Generated. Executing all of our experiments required 48 days of CPU time. The collected data, including the approximated query answers for every scenario with the corresponding running times, amounts to 130GB of uncompressed logs.

7 EXPERIMENTAL EVALUATION

We are now ready to proceed with the evaluation and comparison of the approximation schemes for CQA. As explained in Section 5, the implementation of the approximation schemes for CQA deviates from the pseudocode given in Algorithm 1. We first construct, via a preprocessing step, the set of pairs $\mathsf{syn}_{\Sigma,Q}(D)$ for the given database D, set Σ of primary keys, and CQ Q, and then iterate over the pairs



(a) Noise[0, 1] (b) Noise[0, 3] (c) Noise[0, 5] (d) Noise[0.3, 1] (e) Noise[0.3, 3]

(f) Noise[0.3, 5] (g) Noise[0.5, 1] (h) Noise[0.5, 3] (i) Noise[0.5, 5]

Figure 1: Noise test scenarios - Noise[*balance*, *joins*]

of $\mathsf{syn}_{\Sigma,Q}(D)$ and call the approximation scheme for RelativeFreq in question. Let us first briefly report on how the preprocessing step performs over the database-query pairs of P_{H}.

For the wide variety of database-query pairs (D, Q) that we have generated, $\mathsf{syn}_{\Sigma_{\mathrm{H}},Q}(D)$, which is essentially what we give as an input to the approximation schemes for CQA in our experiments, can be effectively computed via the query rewriting approach mentioned in Section 5. Let us stress that our approach is by no means the ultimate way for computing $\mathsf{syn}_{\Sigma_{\mathrm{H}},Q}(D)$ as one may devise a more sophisticated and optimized procedure. Nevertheless, it turned out that it performs reasonably well in almost all of our scenarios. For most pairs (D, Q) of P_{H} it took between 20 and 30 seconds for the computation of the set $\mathsf{syn}_{\Sigma_{\mathrm{H}},Q}(D)$. More precisely, for 80% of the pairs of P_{H}, the preprocessing step completed its execution in less than 30 seconds, for 94% in less than a minute, while the execution time over all pairs never exceeded two minutes. Although there is room for improvement in terms of the running time of the preprocessing step, the take-home message is that relying on such a preprocessing step is not prohibitive in practice; furthers details can be found in the appendix.

7.1 Evaluating Approximate CQA

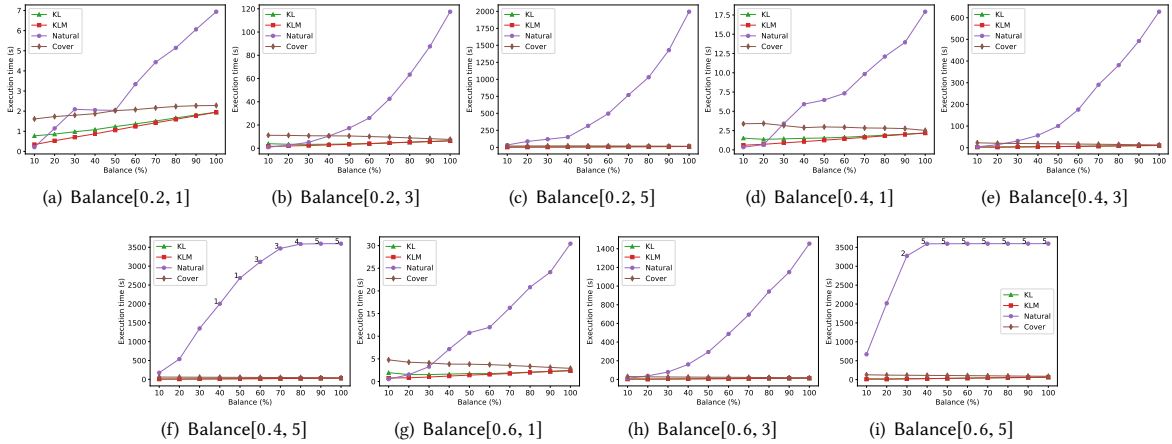
We now focus on the task of evaluating and comparing the main approximation schemes for CQA presented in Section 4. In particular, we consider ApxCQA[Natural], ApxCQA[KL], ApxCQA[KLM], and ApxCQA[Cover]; henceforth, for brevity, we simply write A instead of ApxCQA[A]. Note that the running times that we are going to present do not consider the time of the preprocessing step since it is the same for all the approximation schemes.

Noise Scenarios. Although we performed experiments for all the 55 noise scenarios, we present only the results of nine representative scenarios (see Figure 1); the rest can be found in the appendix. Let us stress, however, that the main conclusions that we draw below from those nine scenarios are in line with what we can conclude from all the 55 noise scenarios. The plots depicted in Figure 1 show how the running time of the approximation schemes for CQA is affected by varying the noise, where for each noise level the running time is the average over all the CQs for that level of noise (five in total). The fact that an approximation scheme timed out for n out of the five CQs for a noise level is indicated by the integer n. It turned out that there is a striking difference on which algorithm performs better depending on whether the CQs are Boolean or not.

The Boolean Case. The running time of the approximation scheme Natural is not significantly affected by the increase of noise whenever we focus on Boolean CQs (see plots (a), (b) and (c) of Figure 1). On the other hand, the running time of KL, KLM and Cover quickly increases as we increase the level of noise. They even timeout for some queries at high levels of noise ($\geq 70\%$) and many joins (≥ 3). Let us explain the reasons behind those observations.

Consider the set $\mathsf{syn}_{\Sigma_{\mathrm{H}},Q}(D)$ over which the algorithms are executed. Natural and KL(M) compute the number of iterations via a call to the optimal estimator OptEstimate[Sample]$((\mathcal{H},\mathcal{B}),\epsilon,\delta)$, for each pair $(\bar{t},(\mathcal{H},\mathcal{B})) \in \mathsf{syn}_{\Sigma_{\mathrm{H}},Q}(D)$, with Sample being the adopted sampler. The runtime of this procedure is proportional to the variance of Sample$((\mathcal{H},\mathcal{B}))$, and to the inverse of the expected value $\mathbb{E}[\mathsf{Sample}((\mathcal{H},\mathcal{B}))]$, which is not surprising since the computed number of iterations is optimal [8]. For Natural, the value $\mathbb{E}[\mathsf{Sample}((\mathcal{H},\mathcal{B}))]$ coincides with $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$, while for KL(M) is $\mathsf{Num}/|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|$, where Num is the numerator of $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$. When focusing on Boolean CQs, $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$ is generally close to one since the set $\mathsf{syn}_{\Sigma_{\mathrm{H}},Q}(D) = \{(\langle\rangle,(\mathcal{H},\mathcal{B}))\}$ is a singleton, and thus, the only synopsis therein collects all the homomorphic images of the query. This implies that Num is close to $|\mathsf{db}(\mathcal{B})|$. On the other hand, the fact that all the homomorphic images are collected in the set $\mathcal{H}$ implies that $|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|$ becomes very large, in particular, much larger than $|\mathsf{db}(\mathcal{B})|$, and the difference increases as we increase the amount of noise. Therefore, $\mathsf{Num}/|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|$, which coincides with $\mathbb{E}[\mathsf{SampleKL}((\mathcal{H},\mathcal{B}))]$ and $\mathbb{E}[\mathsf{SampleKLM}((\mathcal{H},\mathcal{B}))]$, quickly decreases as the noise increases. Moreover, the large size of $\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}$ makes sampling from the symbolic sampling space more demanding as the noise increases. This trend is even more evident when we consider more joins since the size of the set $\mathcal{H}$ tends to be higher.

Concerning the approximation scheme Cover, its high runtime is explained by the fact that its number of iterations is linear in



(a) Balance[0.2, 1] (b) Balance[0.2, 3] (c) Balance[0.2, 5] (d) Balance[0.4, 1] (e) Balance[0.4, 3]

(f) Balance[0.4, 5] (g) Balance[0.6, 1] (h) Balance[0.6, 3] (i) Balance[0.6, 5]

Figure 2: Balance test scenarios - Balance[*noise*, *joins*]

$|\mathcal{H}|$ [15], which, as already discussed, is generally large for Boolean queries. Moreover, as mentioned above, the runtime increase is also explained by the fact that Cover draws samples from the symbolic sampling space $\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}$, which becomes more time consuming as the noise increases. As an additional remark, let us say that Cover, contrary to what one would expect, is the worst performer despite the fact that its running time is linear in $|\mathcal{H}|$, in contrast to KL(M) that rely on a more complex sampler, and thus, they depend quadratically on $|\mathcal{H}|$.[5] The reason for this is that the factor that is multiplied by $|\mathcal{H}|$ in the number of iterations of Cover can become very large, even for not very small values of ϵ and δ such as 0.1 (i.e., 10% error) and 0.25 (i.e., 75% confidence). Hence, the fact that KL(M) performs much less iterations than Cover leads to a better performance despite the fact that SampleKL and SampleKLM are more demanding samplers. One would start observing that Cover performs similarly or better than KL(M) only when $|\mathcal{H}|$ becomes very large, which, in the Boolean case, happens only when we consider many joins, and high levels of noise (see plot (c) in Figure 1).

The Non-Boolean Case. Focusing on non-Boolean CQs (see plots (d), (e) and (f) in Figure 1), Natural is no more among the fastest algorithms. Actually, its runtime rapidly increases as we increase the amount of noise, whereas KL(M) perform much better. The reason for this is that in the case of queries with higher balance, the number of synopses in $\mathsf{syn}_{\Sigma_{\mathrm{H}},Q}(D) = \{(\bar{t}_i, (\mathcal{H}_i, \mathcal{B}_i))\}_{i\in[n]}$ increases. Since $\mathcal{H}_1, \ldots, \mathcal{H}_n$ form a partition of $\bigcup_{i=1}^{n} \mathcal{H}_i$, the cardinality of each $\mathcal{H}_i$ tends to decrease as we increase the balance. Therefore, the expected value $\mathbb{E}[\mathsf{SampleNatural}((\mathcal{H}_i, \mathcal{B}_i))]$, which is equal to $\mathsf{R}_{(\mathcal{H}_i,\mathcal{B}_i)}$, tends to be closer to zero, whereas the values $\mathbb{E}[\mathsf{SampleKL}((\mathcal{H}_i, \mathcal{B}_i))]$ and $\mathbb{E}[\mathsf{SampleKLM}((\mathcal{H}_i, \mathcal{B}_i))]$, which coincide, are closer to one. We can then explain the runtime of Natural and KL(M) by providing a discussion similar to the one above for the Boolean case. Moreover, Cover is now considerably faster since the cardinality of each $\mathcal{H}_i$ tends to be small, and thus, it performs much less iterations. However, due to the reason discussed above in the Boolean case, it still performs worse than KL(M). The behaviour described above becomes even more evident for CQs with even higher balance (see plots (g), (h) and (i) in Figure 1).

Balance Scenarios. As we have already observed, the balance of the query has an effect on the running time of the approximation schemes for CQA. To quantify the impact of the balance on the running time, we consider here the family of balance scenarios. We ran experiments for all the scenarios but we present only the results of nine representative ones (see Figure 2); the rest can be found in the appendix. Let us note, however, that the main conclusions that we draw below are in line with what we can conclude from all the 50 balance scenarios. Figure 2 shows how the running time of the approximation schemes for CQA is affected by varying the balance of the query, where for each balance level the running time is the average over all CQs with that level of balance (five in total).

It is clear that Natural is the worst performer with timeouts at high number of joins and high levels of noise. On the other hand, KL and KLM perform better followed by Cover. Another interesting observation is that Cover is, in general, the only approximation scheme that its running time decreases as the balance increases. Let us explain the reasons behind those observations.

Concerning Natural, the plots depicted in Figure 2 confirm what we discussed above in the analysis of the noise scenarios: having more synopses $(\mathcal{H}, \mathcal{B})$ in $\mathsf{syn}_{\Sigma_{\mathrm{H}},Q}(D)$, we get a lower ratio $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$, which in turn leads to higher running time for a single synopsis. In fact, for queries with low balance, Natural is usually comparable to the other algorithms, unless we consider a high level of noise and many joins, which also contribute in decreasing $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$.

Regarding the approximation schemes KL and KLM, as discussed in the analysis of the noise scenarios, higher balance means that the values $\mathbb{E}[\mathsf{SampleKL}((\mathcal{H}, \mathcal{B}))]$ and $\mathbb{E}[\mathsf{SampleKLM}((\mathcal{H}, \mathcal{B}))]$, which coincide, are higher. Hence, the running time for a single synopsis decreases. However, higher balance also means that there are more synopses in $\mathsf{syn}_{\Sigma_{\mathrm{H}},Q}(D)$ that must be processed by the approximation scheme. After some level of balance, there is essentially no improvement of the running time for a single synopsis, as the number of iterations computed by OptEstimate is as small as it can get. This is confirmed by the fact that for the very first levels of

[5] Since $\mathbb{E}[\mathsf{SampleKL}((\mathcal{H}, \mathcal{B}))] = \mathbb{E}[\mathsf{SampleKLM}((\mathcal{H}, \mathcal{B}))]$ is greater or equal to $1/|\mathcal{H}|$, and the samplers SampleKL(M) need to iterate over every index in $[|\mathcal{H}|]$.

balance, KL(M) decrease in runtime, but after that, the running time is essentially dominated by the number of synopses over which the algorithms need to iterate, and thus, it increases linearly.

On the contrary, Cover always benefits from an increase in balance. This is due to the fact that the number of iterations that Cover has to perform keeps decreasing as the balance increases (since $|\mathcal{H}|$ decreases). However, although the number of iterations always decreases, this remains considerably higher than the optimal number of iterations of KL(M), due to the high constant factors. Thus, Cover performs worse than KL(M) in terms of plain running time.

Join Scenarios. It turned out that the experimental analysis based on the join scenarios did not provide any new insights on the behaviour of the approximation schemes; the analysis can be found in the appendix. There is an interesting observation, though, which cannot be readily seen from the analysis based on the noise and balance scenarios, and concerns the comparison between KL and KLM. Although for a small number of joins KLM performs better than KL (for any level of noise), as we increase the number of joins, KL catches up, and in the Boolean case may even perform better.

7.2 Take-home Messages

Here are the take-home messages of our analysis, which reveal a striking difference between Boolean and non-Boolean CQs. A more detailed discussion can be found in the appendix.

(1) For Boolean CQs, Natural is the best performer, no matter the amount of noise, and no matter the number of joins in the query, whereas Cover is the worst. Only in the case of CQs with many joins Cover is comparable to KL(M), but in any case, Natural is the indicated scheme. Interestingly, this outcome is in contrast to what is generally believed in the DNF setting, where sampling from the natural sampling space is regarded to be impractical [6, 15].

(2) For non-Boolean CQs, KLM is the way to go in almost all the scenarios, i.e., for any level of noise and for any level of (non-zero) balance of the query. Only for CQs with many joins KL is comparable to KLM. Nevertheless, KL is never going to outperform KLM. The worst algorithms are Natural and Cover. They perform similarly for low levels of noise and balance (regardless the number of joins), but, in general, Natural is the slowest.

(3) We can safely claim that approximate CQA in the presence of primary keys is feasible in practice. We have seen that the preprocessing step, which is responsible for computing the synopses, has completed its execution in less than 30 seconds in most cases. Furthermore, for modest scenarios, which is what we expect to face in practice, the running time of the best performing approximation scheme is reasonable; e.g., for 50% noise, 50% balance, and CQs with 3 joins, the runtime of the best performer is at most 6 seconds, and this decreases to 3 seconds for 30% noise, and 30% balance.

Let us conclude this section by stressing that we have also experimentally validated our main conclusions concerning the main approximation schemes for CQA (take-home messages (1) and (2)) by relying on a different batch of test scenarios that are closer to real-world use cases. The details can be found in the appendix.

8 CONCLUSIONS

Our work provides the first comprehensive and publicly available benchmark (test infrastructure and test scenarios) for randomized approximation schemes for CQs and primary keys. It also provides experimental results for the main data-efficient randomized approximation schemes that can be inherited from the DNF literature, and adapted to the CQA setting, with new insights on which approximation technique is indicated depending on characteristics of the input. Our intention is to maintain and enrich our test infrastructure, as well as our (test and validation) scenarios.

REFERENCES

[1] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.

[2] Patricia C. Arocena, Boris Glavic, Giansalvatore Mecca, Renée J. Miller, Paolo Papotti, and Donatello Santoro. 2015. Messing Up with BART: Error Generation for Evaluating Data-Cleaning Algorithms. *PVLDB* 9, 2 (2015), 36–47.

[3] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. 2017. Benchmarking the Chase. In *PODS*. 37–52.

[4] George Beskales, Ihab F. Ilyas, and Lukasz Golab. 2010. Sampling the Repairs of Functional Dependency Violations under Hard Constraints. *PVLDB* 3, 1 (2010), 197–207.

[5] Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. 2005. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *SIGMOD*. 143–154.

[6] Marco Calautti, Marco Console, and Andreas Pieris. 2019. Counting Database Repairs under Primary Keys Revisited. In *PODS*. 104–118.

[7] Marco Calautti, Leonid Libkin, and Andreas Pieris. 2018. An Operational Approach to Consistent Query Answering. In *PODS*. 239–251.

[8] Paul Dagum, Richard M. Karp, Michael Luby, and Sheldon M. Ross. 2000. An Optimal Algorithm for Monte Carlo Estimation. *SIAM J. Comput.* 29, 5 (2000), 1484–1496.

[9] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: A commodity data cleaning system. In *SIGMOD*. 541–552.

[10] Nilesh N. Dalvi and Dan Suciu. 2007. Management of probabilistic data: foundations and challenges. In *PODS*. 1–12.

[11] Akhil A. Dixit and Phokion G. Kolaitis. 2019. A SAT-Based System for Consistent Query Answering. In *SAT*. 117–135.

[12] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33, 2 (2008), 6:1–6:48.

[13] Ariel Fuxman, Elham Fazli, and Renée J. Miller. 2005. ConQuer: Efficient Management of Inconsistent Databases. In *SIGMOD*. 155–166.

[14] Richard M. Karp and Michael Luby. 1983. Monte-Carlo Algorithms for Enumeration and Reliability Problems. In *FOCS*. 56–64.

[15] Richard M. Karp, Michael Luby, and Neal Madras. 1989. Monte-Carlo Approximation Algorithms for Enumeration Problems. *J. Algorithms* 10, 3 (1989), 429–448.

[16] Phokion G. Kolaitis, Enela Pema, and Wang-Chiew Tan. 2013. Efficient Querying of Inconsistent Databases with Binary Integer Programming. *PVLDB* 6, 6 (2013), 397–408.

[17] Paraschos Koutris and Dan Suciu. 2014. A Dichotomy on the Complexity of Consistent Query Answering for Atoms with Simple Keys. In *ICDT*. 165–176.

[18] Paraschos Koutris and Jef Wijsen. 2015. The Data Complexity of Consistent Query Answering for Self-Join-Free Conjunctive Queries Under Primary Key Constraints. In *PODS*. 17–29.

[19] Paraschos Koutris and Jef Wijsen. 2019. Consistent Query Answering for Primary Keys in Logspace. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*. 23:1–23:19.

[20] Paraschos Koutris and Jef Wijsen. 2020. First-Order Rewritability in Consistent Query Answering with Respect to Multiple Keys. In *PODS*. 113–129.

[21] Dany Maslowski and Jef Wijsen. 2013. A dichotomy in the complexity of counting database repairs. *J. Comput. Syst. Sci.* 79, 6 (2013), 958–983.

[22] Dany Maslowski and Jef Wijsen. 2014. Counting Database Repairs that Satisfy Conjunctive Queries with Self-Joins. In *ICDT*. 155–164.

[23] Makoto Matsumoto and Takuji Nishimura. 1998. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Trans. Model. Comput. Simul.* 8, 1 (1998), 3–30.

[24] Kuldeep S. Meel, Aditya A. Shrotri, and Moshe Y. Vardi. 2019. Not all FPRASs are equal: Demystifying FPRASs for DNF-counting. *Constraints* 24, 3-4 (2019), 211–233.

[25] Balder ten Cate, Gaëlle Fontaine, and Phokion G. Kolaitis. 2015. On the Data Complexity of Consistent Query Answering. *Theory Comput. Syst.* 57, 4 (2015), 843–891.

[26] Vijay V. Vazirani. 2003. *Approximation Algorithms*.

A NOTIONS FROM PROBABILITY THEORY

A *(discrete) probability space* is a pair $\mathsf{PS} = (\Omega, \pi)$, where Ω is a finite set, called *sample space*, and $\pi : \Omega \to [0,1]$ is a function such that $\sum_{\omega \in \Omega} \pi(\omega) = 1$. A subset $E \subseteq \Omega$ is called an *event*. The probability of an event E, denoted $\Pr(E)$ is defined as $\sum_{\omega \in E} \pi(\omega)$. A *random variable* over PS is a function $X : \Omega \to \mathbb{Q}$. For every $x \in \mathbb{Q}$, $X = x$ denotes the event $\{\omega \in \Omega \mid X(\omega) = x\}$. More complex events involving inequalities and other constraints are defined in a similar way. The *expected value* of X, denoted $\mathbb{E}[X]$ is defined as

$$\sum_{x \in X(\Omega)} x \cdot \Pr(X = x),$$

where $X(\Omega)$ is the image of X, that is,

$$X(\Omega) \;=\; \{x \in \mathbb{Q} \mid \text{ there exists } \omega \in \Omega \text{ such that } X(\omega) = x\}.$$

Let $X_1, \ldots, X_n$ be random variables over some probability space $\mathsf{PS} = (\Omega, \pi)$. We say that $X_1, \ldots, X_n$ are *independent* if

$$\Pr\left(\bigcap_{i=1}^{n} X_i = x_i\right) \;=\; \prod_{i=1}^{n} \Pr(X_i = x_i),$$

for all $x_1, \ldots, x_n \in \mathbb{Q}$. If $X_1, \ldots, X_n$ are independent and $X_i(\Omega) \subseteq [0,1]$, for each $i \in [n]$, then Hoeffding's inequality states that

$$\Pr(|\bar{X}_n - \mathbb{E}[\bar{X}_n]| \leq \gamma) \;\geq\; 1 - 2e^{-2n\gamma^2},$$

for every $\gamma > 0$, where $\bar{X}_n$ denotes the random mean of the random variables $X_1, \ldots, X_n$, i.e., the random variable over (Ω, π), such that

$$\bar{X}_n(\omega) \;=\; \frac{1}{n} \cdot \sum_{i=1}^{n} X_i(\omega),$$

for each $\omega \in \Omega$. Note that if all X_i's have the same mean μ, then $\Pr(|\bar{X}_n - \mu| \leq \gamma) \geq 1 - 2e^{-2n\gamma^2}$, for every $\gamma > 0$.

B APPROXIMATION SCHEMES

In this section, we provide proofs for the technical results given in Section 4, as well as details concerning the self-adjusting coverage algorithm that are missing from the main body of the paper.

Proof of Lemma 4.1

Item (1): There exist at most polynomially many (w.r.t. $||D||$ and $||\bar{t}||$) homomorphisms from $Q(\bar{x})$ to D with $h(\bar{x}) = \bar{t}$. Thus, to construct $\mathcal{H}$, it suffices to iterate over each such homomorphism h, construct $h(Q)$, and check whether $h(Q) \models \Sigma$, in which case $h(Q)$ is added to $\mathcal{H}$. Once we have $\mathcal{H}$ in place, we can construct $\mathcal{B}$ via a straightforward search over D.

Item (2): It follows by definition of (Σ, Q)-synopsis.

Item (3): Consider the set of homomorphisms

$$H \;=\; \{h \mid h \text{ is a homomorphism from } Q(\bar{x}) \text{ to } D \text{ with } h(\bar{x}) = \bar{t}, \text{ and } h(Q) \models \Sigma\}.$$

Therefore, the numerator Num of $\mathsf{R}_{D,\Sigma,Q}(\bar{t})$ is equal to

$$|\{D' \in \mathsf{rep}(D,\Sigma) \mid \text{there exists a homomorphism } h \in H \text{ such that } h(Q) \subseteq D'\}|.$$

Consider now the partition $\{D^+, D^-\}$ of D, where D^+ contains all the blocks of facts occurring in $\bigcup_{h \in H} h(Q)$, and D^- contains the remaining blocks. Consequently, Num is equal to

$$|\{D' \in \mathsf{rep}(D^+,\Sigma) \mid \text{there exists a homomorphism } h \in H \text{ such that } h(Q) \subseteq D'\}| \cdot |\mathsf{rep}(D^-,\Sigma)|.$$

The denominator of $\mathsf{R}_{D,\Sigma,Q}(\bar{t})$, i.e., $|\mathsf{rep}(D,\Sigma)|$, is equal to

$$|\mathsf{rep}(D^+,\Sigma)| \cdot |\mathsf{rep}(D^-,\Sigma)|.$$

Hence,

$$\mathsf{R}_{D,\Sigma,Q}(\bar{t}) \;=\; \frac{|\{D' \in \mathsf{rep}(D^+,\Sigma) \mid \exists h \in H \text{ s.t. } h(Q) \subseteq D'\}|}{|\mathsf{rep}(D^+,\Sigma)|}.$$

Since D^+ coincides with $\mathcal{B}$, the above expression is precisely the ratio $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$, and the claim follows.

Item (4): Assume that $\mathcal{H} \neq \emptyset$. Then, there exists a homomorphism h from $Q(\bar{x})$ to D, where $h(\bar{x}) = \bar{t}$ and $h(Q) \models \Sigma$. Clearly, there exists a repair $D' \in \mathsf{rep}(D,\Sigma)$ that contains $h(Q)$, and thus $\bar{t} \in Q(D')$, which in turn implies that $\mathsf{R}_{D,\Sigma,Q}(\bar{t}) \neq 0$.

Conversely, assume that $\mathsf{R}_{D,\Sigma,Q}(\bar{t}) \neq 0$, and let $D' \in \mathsf{rep}(D,\Sigma)$ be the repair such that $\bar{t} \in Q(D')$. The latter is equivalent to the fact that there exists a homomorphism h from $Q(\bar{x})$ to D' with $h(\bar{x}) = \bar{t}$. Since $D' \subseteq D$, h is also a homomorphism from Q to D. Since $D' \models \Sigma$, we get that $h(Q) \models \Sigma$, and therefore $\mathcal{H} \neq \emptyset$, as needed.

Proof of Lemma 4.3

Consider an admissible pair $(\mathcal{H}, \mathcal{B})$. The fact that $\mathsf{SampleNatural}$ runs in polynomial time w.r.t. $||\mathcal{H}, \mathcal{B}||$ is straightforward. We now consider the expected value of $\mathsf{SampleNatural}((\mathcal{H}, \mathcal{B}))$. Note that $X = \mathsf{SampleNatural}((\mathcal{H}, \mathcal{B}))$ is the random variable from the probability space (Ω, π) to $\{0,1\}$, where $\Omega = \mathcal{S}_{(\mathcal{H},\mathcal{B})}$ and π is the uniform distribution over $\mathcal{S}_{(\mathcal{H},\mathcal{B})}$, i.e., $\pi(I) = 1/|\mathcal{S}_{(\mathcal{H},\mathcal{B})}|$, for each $I \in \mathcal{S}_{(\mathcal{H},\mathcal{B})}$. Moreover, $X(I) = 1$, whenever there exists $H \in \mathcal{H}$ such that $H \subseteq I$, and 0 otherwise. Therefore,

$$\Pr(X = 1) \;=\; \sum_{\{I \in \Omega \mid \exists H \in \mathcal{H} \text{ s.t. } H \subseteq I\}} \pi(I) = \mathsf{R}_{(\mathcal{H},\mathcal{B})}.$$

Thus, $\mathbb{E}[X] = \mathbb{E}[\mathsf{SampleNatural}((\mathcal{H}, \mathcal{B}))] = \mathsf{R}_{(\mathcal{H},\mathcal{B})}$.

It remains to show that when $\mathbb{E}[\mathsf{SampleNatural}((\mathcal{H}, \mathcal{B}))] > 0$, then $\mathbb{E}[\mathsf{SampleNatural}((\mathcal{H}, \mathcal{B}))] \geq 1/\mathrm{p}(||\mathcal{H}, \mathcal{B}||)$, for some polynomial p. From the above discussion, this is equivalent to show that $\mathsf{R}_{(\mathcal{H},\mathcal{B})} > 0$ implies $\mathsf{R}_{(\mathcal{H},\mathcal{B})} \geq 1/\mathrm{p}(||\mathcal{H}, \mathcal{B}||)$. If $\mathsf{R}_{(\mathcal{H},\mathcal{B})} > 0$, then there exists $I \in \mathsf{db}(\mathcal{B}) = \mathcal{S}_{(\mathcal{H},\mathcal{B})}$ such that $H \subseteq I$, for some $H \in \mathcal{H}$. Fix such an H, and let $\mathcal{B}_H$ be the set of all blocks in $\mathcal{B}$ with key values occurring in H. Then, every $I \in \mathsf{db}(\mathcal{B})$ such that $H \subseteq I$ is of the form $H \cup B$, where B is any set from $\mathsf{db}(\mathcal{B} \setminus \mathcal{B}_H)$. Hence, there exist at least $|\mathsf{db}(\mathcal{B} \setminus \mathcal{B}_H)|$ sets $I \in \mathsf{db}(\mathcal{B})$ such that $H' \subseteq I$, for some $H' \in \mathcal{H}$. Consequently,

$$\mathsf{R}_{(\mathcal{H},\mathcal{B})} \;\geq\; \frac{|\mathsf{db}(\mathcal{B} \setminus \mathcal{B}_H)|}{|\mathsf{db}(\mathcal{B})|}.$$

Since $|\mathsf{db}(\mathcal{B})| = |\mathsf{db}(\mathcal{B}_H)| \cdot |\mathsf{db}(\mathcal{B} \setminus \mathcal{B}_H)|$, we conclude that

$$\mathsf{R}_{(\mathcal{H},\mathcal{B})} \;\geq\; \frac{1}{|\mathsf{db}(\mathcal{B}_H)|} \;\geq\; \frac{1}{b^h},$$

where b is the size of the largest $B \in \mathsf{db}(\mathcal{B})$ and h is the size of the largest $H \in \mathcal{H}$. Clearly, b^h is a polynomial of $||\mathcal{H}, \mathcal{B}||$.

Proof of Lemma 4.5

Let $(\mathcal{H}, \mathcal{B})$ be an admissible pair, with $\mathcal{H} = \{H_1, \ldots, H_n\}$. To show that SampleKL$((\mathcal{H}, \mathcal{B}))$ runs in polynomial time w.r.t. $||\mathcal{H}, \mathcal{B}||$, it suffices to show that a pair $(i, I) \in \mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}$ can be chosen with probability $1/|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|$ in polynomial time w.r.t. $||\mathcal{H}, \mathcal{B}||$, and that we can check whether $I \in \mathcal{I}^{i}_{(\mathcal{H},\mathcal{B})}$ in polynomial time w.r.t. $||\mathcal{H}, \mathcal{B}||$. For the former task, it suffices to first choose $i \in [n]$ with probability $|\mathcal{I}^{i}_{(\mathcal{H},\mathcal{B})}|/|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|$, and then choose $I \in \mathcal{I}^{i}_{(\mathcal{H},\mathcal{B})}$ with probability $1/|\mathcal{I}^{i}_{(\mathcal{H},\mathcal{B})}|$. For the latter task, it suffices to check whether $H_i \subseteq I$. Observe that the above two tasks can be performed in polynomial time w.r.t. $||\mathcal{H}, \mathcal{B}||$ if we can compute $|\mathcal{I}^{i}_{(\mathcal{H},\mathcal{B})}|$ (for some given $i \in [n]$) and $|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|$ in polynomial time. Since

$$\mathcal{I}^{i}_{(\mathcal{H},\mathcal{B})} \;=\; \{I \in \mathsf{db}(\mathcal{B}) \mid H_i \subseteq I\},$$

we conclude that $|\mathcal{I}^{i}_{(\mathcal{H},\mathcal{B})}| = |\mathsf{db}(\mathcal{B} \setminus \mathcal{B}_{H_i})|$, where $\mathcal{B}_{H_i}$ is the set of blocks in $\mathcal{B}$ with key values in H_i. Computing $|\mathsf{db}(\mathcal{B} \setminus \mathcal{B}_{H_i})|$ is straightforward. To compute $|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|$, observe that

$$|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}| \;=\; \sum_{i=1}^{n} \left|\mathcal{I}^{i}_{(\mathcal{H},\mathcal{B})}\right|.$$

Since each term of the summation can be computed in polynomial time w.r.t. $||\mathcal{H}, \mathcal{B}||$, $|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|$ can be compute in polynomial time.

Let us now focus on the expected value of SampleKL$((\mathcal{H}, \mathcal{B}))$. Note that $X = \mathsf{SampleKL}((\mathcal{H}, \mathcal{B}))$ is a random variable from the probability space (Ω, π) to $\{0, 1\}$, where $\Omega = \mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}$, and π is the uniform distribution. Moreover, for each $(i, I) \in \Omega$,

$$X((i,I)) = \begin{cases} 1 & \text{if there is no } j < i \text{ such that } I \in \mathcal{I}^{j}_{(\mathcal{H},\mathcal{B})}, \\ 0 & \text{otherwise.} \end{cases}$$

Therefore,

$$\Pr(X = 1) \;=\; \sum_{\{(i,I)\in\Omega \mid \nexists j<i \text{ s.t. } I\in\mathcal{I}^{j}_{(\mathcal{H},\mathcal{B})}\}} \pi((i,I)) \;=\; \frac{\mathsf{Num}}{|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|},$$

where Num is the numerator of $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$. Hence,

$$\mathbb{E}[X] \;=\; \mathbb{E}[\mathsf{SampleKL}((\mathcal{H}, \mathcal{B}))] \;=\; \mathsf{R}_{(\mathcal{H},\mathcal{B})} \cdot \frac{|\mathsf{db}(\mathcal{B})|}{|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|}.$$

Recall that $|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|$ can be computed in polynomial time in $||\mathcal{H}, \mathcal{B}||$. It remains to show that $\mathbb{E}[\mathsf{SampleKL}((\mathcal{H}, \mathcal{B}))] > 0$ implies $\mathbb{E}[\mathsf{SampleKL}((\mathcal{H}, \mathcal{B}))] \geq 1/\mathsf{p}(||\mathcal{H}, \mathcal{B}||)$, for some polynomial p. From the above discussion, we get that $\mathbb{E}[\mathsf{SampleKL}((\mathcal{H}, \mathcal{B}))] = \mathsf{Num}/|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|$, where Num is the numerator of $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$. By definition of $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$, $\mathsf{Num} = |\{I \in \mathsf{db}(\mathcal{B}) \mid H_i \subseteq I \text{ for some } H_i \in \mathcal{H}\}|$. Hence, if we let $k \in [n]$ be such that $|\mathcal{I}^{k}_{(\mathcal{H},\mathcal{B})}|$ is maximum,

$$\mathsf{Num} \;\geq\; |\{I \in \mathsf{db}(\mathcal{B}) \mid H_k \subseteq I\}| \;=\; |\mathcal{I}^{k}_{(\mathcal{H},\mathcal{B})}|.$$

Therefore,

$$\mathbb{E}[\mathsf{SampleKL}((\mathcal{H}, \mathcal{B}))] \;\geq\; \frac{|\mathcal{I}^{k}_{(\mathcal{H},\mathcal{B})}|}{|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|} \;\geq\; \frac{|\mathcal{I}^{k}_{(\mathcal{H},\mathcal{B})}|}{n \cdot |\mathcal{I}^{k}_{(\mathcal{H},\mathcal{B})}|} \;=\; \frac{1}{n}.$$

Proof of Lemma 4.7

Consider an admissible pair $(\mathcal{H}, \mathcal{B})$, with $\mathcal{H} = \{H_1, \ldots, H_n\}$. The analysis of the running time of SampleKLM with input $(\mathcal{H}, \mathcal{B})$ is along the lines of the one given in the proof of Lemma 4.5. Concerning the expected value, $X = \mathsf{SampleKLM}((\mathcal{H}, \mathcal{B}))$ is a random variable from the probability space (Ω, π), where $\Omega = \mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}$, and π is the uniform distribution. In this case, the range of X is the set $\{k^{-1}\}_{k\in[n]}$. In particular, for each $(i, I) \in \Omega$, we have that

$$X((i,I)) \;=\; \frac{1}{|\{j \in [n] \mid I \in \mathcal{I}^{j}_{(\mathcal{H},\mathcal{B})}\}|}.$$

Thus, for every $k \in [n]$, we have that

$$\Pr\left(X = \frac{1}{k}\right) \;=\; \sum_{\{(i,I)\in\Omega \;\mid\; |\{j\in[n]|I\in\mathcal{I}^{j}_{(\mathcal{H},\mathcal{B})}\}|=k\}} \pi((i,I)).$$

Consider now a pair $(i, I) \in \Omega$. If I is such that $|\{j \in [n] \mid I \in \mathcal{I}^{j}_{(\mathcal{H},\mathcal{B})}\}| = k$, for some $k \in [n]$, then there exist exactly $k - 1$ distinct pairs $(j_1, I), \ldots, (j_{k-1}, I) \in \Omega$, which are different than (i, I), with the same property. Hence, we conclude that

$$\Pr\left(X = \frac{1}{k}\right) \;=\; k \cdot \frac{\mathsf{Num}_k}{|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|},$$

where Num_k is the number of all $I \in \mathsf{db}(\mathcal{B})$ such that $|\{j \in [n] \mid I \in \mathcal{I}^{j}_{(\mathcal{H},\mathcal{B})}\}| = k$. The latter is equivalent to say that Num_k is the number of all $I \in \mathsf{db}(\mathcal{B})$ that contain exactly k distinct sets in $\mathcal{H}$. Thus, the expected value of $X = \mathsf{SampleKLM}((\mathcal{H}, \mathcal{B}))$ is

$$\mathbb{E}[X] \;=\; \sum_{k=1}^{n} \frac{1}{k} \cdot \Pr\left(X = \frac{1}{k}\right) \;=\; \sum_{k=1}^{n} \frac{\mathsf{Num}_k}{|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|}.$$

Clearly, $\sum_{k=1}^{n} \mathsf{Num}_k = \mathsf{Num}$, where Num is the numerator of $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$, which in turn implies that

$$\mathbb{E}[X] \;=\; \mathbb{E}[\mathsf{SampleKLM}((\mathcal{H}, \mathcal{B}))] \;=\; \frac{\mathsf{Num}}{|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|}.$$

Therefore,

$$\mathbb{E}[\mathsf{SampleKLM}((\mathcal{H}, \mathcal{B}))] \;=\; \mathsf{R}_{(\mathcal{H},\mathcal{B})} \cdot \frac{\mathsf{db}(\mathcal{B})}{|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|}.$$

We have already shown that computing $|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|$ is feasible in polynomial time w.r.t. $||\mathcal{H}, \mathcal{B}||$. Moreover, from Lemma 4.5, we know that there exists a polynomial p such that $\mathbb{E}[\mathsf{SampleKL}((\mathcal{H}, \mathcal{B}))] = \mathbb{E}[\mathsf{SampleKLM}((\mathcal{H}, \mathcal{B}))] > 0$ implies $\mathbb{E}[\mathsf{SampleKL}((\mathcal{H}, \mathcal{B}))] = \mathbb{E}[\mathsf{SampleKLM}((\mathcal{H}, \mathcal{B}))] \geq 1/\mathsf{p}(||\mathcal{H}, \mathcal{B}||)$, and the claim follows.

Self-adjusting Approximation

The SelfAdjustingCoverage algorithm, which is an adaptation of an algorithm given in [15], is depicted in Algorithm 6. We need to show that SelfAdjustingCoverage is an efficient approximation scheme for UnionOfSets, which will immediately imply Theorem 4.9.

Our adapted algorithm essentially takes as input an admissible pair $(\mathcal{H}, \mathcal{B})$, where $\mathcal{H} = \{H_1, \ldots, H_n\}$, as the description of the sets $\mathcal{I}^{1}_{(\mathcal{H},\mathcal{B})}, \ldots, \mathcal{I}^{n}_{(\mathcal{H},\mathcal{B})}$. Thus, from results in [15], for showing that SelfAdjustingCoverage is an efficient approximation scheme for UnionOfSets, it suffices to show that the following tasks can be carried out in polynomial time w.r.t. $||\mathcal{H}, \mathcal{B}||$:

Input: An admissible pair $(\mathcal{H}, \mathcal{B})$, $\epsilon > 0$, and $0 < \delta < 1$
Output: A random number in $[0, 1]$

$steps := 0$; $total := 0$; $trials := 0$
$N := \lceil (8 \cdot (1 + \epsilon) \cdot |\mathcal{H}| \cdot \ln(3/\delta))/((1 - \epsilon^2/8) \cdot \epsilon^2) \rceil$
while true **do**
  Choose $(i, I) \in \mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}$ with probability $\frac{1}{|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|}$
  while true **do**
    $steps := steps + 1$
    if $steps > N$ **then**
      goto finish
    Choose $j \in \{1, \ldots, |\mathcal{H}|\}$ with probability $\frac{1}{|\mathcal{H}|}$
    if $I \in \mathcal{I}^{j}_{(\mathcal{H},\mathcal{B})}$ **then**
      goto end_inner_while
  end_inner_while:
  $total := steps$
  $trials := trials + 1$
finish:
$p := \frac{total \cdot |\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|}{|\mathcal{H}| \cdot trials}$
return p

Algorithm 6: SelfAdjustingCoverage

(1) choose a pair $(i, I) \in \mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}$ with probability $1/|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|$,
(2) choose a number $j \in \{1, \ldots, |\mathcal{H}|\}$ with probability $1/|\mathcal{H}|$,
(3) check whether $I \in \mathcal{I}^{i}_{(\mathcal{H},\mathcal{B})}$, for some $i \in [n]$, and
(4) compute the number $|\mathcal{S}^{\bullet}_{(\mathcal{H},\mathcal{B})}|$.

We have already shown in the proof of Lemma 4.5 that tasks 1, 3 and 4 can be carried out in polynomial time w.r.t. $||\mathcal{H}, \mathcal{B}||$. Moreover, choosing $j \in \{1, \ldots, |\mathcal{H}|\}$ with probability $1/|\mathcal{H}|$ is clearly feasible in polynomial time w.r.t. $||\mathcal{H}, \mathcal{B}||$, and the claim follows.

C THE PREPROCESSING STEP

We start by making a simple but useful observation. All the algorithms in Section 4, as well as Algorithm 6, that take as input an admissible pair $(\mathcal{H}, \mathcal{B})$, which is essentially the (Σ, Q)-synopsis of D for some tuple $\bar{t}$, are oblivious to the syntactic shape of the facts in $\mathcal{H}$ and $\mathcal{B}$, i.e., the actual relation and tuple of constants of those facts is irrelevant to the execution of the algorithms. This allows us to work with an encoding of $\mathsf{syn}_{\Sigma,Q}(D)$, denoted $\mathsf{enc}(\mathsf{syn}_{\Sigma,Q}(D))$, using integer identifiers for facts, which can be easily constructed by first executing a simple rewriting Q^{rew} of Q over D, and then construct $\mathsf{enc}(\mathsf{syn}_{\Sigma,Q}(D))$ from $Q^{\mathsf{rew}}(D)$ in linear time in $|Q^{\mathsf{rew}}(D)|$.

Computing the Encoding of the Synopsis Set

Assume that the given CQ $Q(\bar{x})$ is of the form $\exists \bar{y}\,(R_1(\bar{z}_1) \wedge \cdots \wedge R_n(\bar{z}_n))$. Suppose, for the moment, that we have a rewriting Q^{rew} of Q of arity $(|\bar{x}| + 4 \cdot n)$ such that $Q^{\mathsf{rew}}(D)$ is the set of tuples

$$\{(h(\bar{x}), \mathsf{rid}_1, \mathsf{bid}_1, \mathsf{tid}_1, \mathsf{kcnt}_1, \ldots, \mathsf{rid}_n, \mathsf{bid}_n, \mathsf{tid}_n, \mathsf{kctr}_n) \mid h \text{ is a homomorphism from } Q \text{ to } D\},$$

where, for each $i \in [n]$, rid_i (relation id) is an integer that identifies the relation R_i, bid_i (block id) is an integer that identifies the block $\mathsf{block}_\Sigma(R_i(h(\bar{z}_i)), D_{|R_i})$, with $D_{|R_i}$ being the restriction of D to the relation R_i, tid_i (tuple id) is an integer that identifies the fact $R_i(h(\bar{z}_i))$ in $\mathsf{block}_\Sigma(R_i(h(\bar{z}_i)), D_{|R_i})$, and kcnt_i (key count) is the cardinality of $\mathsf{block}_\Sigma(R_i(h(\bar{z}_i)), D)$. Observe that the pair $(\mathsf{rid}_i, \mathsf{bid}_i)$ uniquely determines the block $\mathsf{block}_\Sigma(R_i(h(\bar{z}_i)), D) \in \mathsf{block}_\Sigma(D)$, while the triple $(\mathsf{rid}_i, \mathsf{bid}_i, \mathsf{tid}_i)$ uniquely determines the fact $R_i(h(\bar{z}_i)) \in D$. In what follows, we write $[\![\mathsf{rid}_i, \mathsf{bid}_i]\!]$ and $[\![\mathsf{rid}_i, \mathsf{bid}_i, \mathsf{tid}_i]\!]$ for the integer identifier of $\mathsf{block}_\Sigma(R_i(h(\bar{z}_i)), D)$ and $R_i(h(\bar{z}_i))$, respectively, and we will refer to blocks and facts using their integer identifiers. It should be clear that given a tuple

$$(h(\bar{x}), \mathsf{rid}_1, \mathsf{bid}_1, \mathsf{tid}_1, \mathsf{kcnt}_1, \ldots, \mathsf{rid}_n, \mathsf{bid}_n, \mathsf{tid}_n, \mathsf{kctr}_n) \quad (1)$$

of $Q^{\mathsf{rew}}(D)$, the set $H = \{[\![\mathsf{rid}_i, \mathsf{bid}_i, \mathsf{tid}_i]\!]\}_{i \in [n]}$ is the homomorphic image of Q in D via the homomorphism h. Moreover, $H \models \Sigma$ iff for each $i, j \in [n]$, with $i \neq j$, $(\mathsf{tbl}_i, \mathsf{bid}_i) = (\mathsf{tbl}_j, \mathsf{bid}_j)$ implies $\mathsf{tid}_i = \mathsf{tid}_j$. Hence, $\mathsf{enc}(\mathsf{syn}_{\Sigma,Q}(D))$ can be constructed by iterating over the tuples of $Q^{\mathsf{rew}}(D)$, and for each tuple of the form (1), if $H = \{[\![\mathsf{rid}_i, \mathsf{bid}_i, \mathsf{tid}_i]\!]\}_{i \in [n]} \models \Sigma$, then H is added to the $\mathcal{H}$-component of the (Σ, Q)-synopsis of D for $h(\bar{x})$, and, for each $i \in [n]$, the fact $[\![\mathsf{rid}_i, \mathsf{bid}_i, \mathsf{tid}_i]\!]$ is added to the block $[\![\mathsf{rid}_i, \mathsf{bid}_i]\!]$ of the $\mathcal{B}$-component of the (Σ, Q)-synopsis of D for $h(\bar{x})$.

After the completion of the above procedure, some identifiers might be missing from the blocks of the $\mathcal{B}$-component; in fact, the identifiers that do not occur in a set of the $\mathcal{H}$-component. However, since we know the cardinality of each block of the $\mathcal{B}$-component, we simply need to add the integer identifiers that are missing, which in turn leads to the set $\mathsf{enc}(\mathsf{syn}_{\Sigma,Q}(D))$. It remains to explain how the rewriting Q^{rew} of Q is defined as an SQL query.

The Rewriting

For a relation R/n, we define a query Q_R of arity $n + 4$ such that $Q_R(D)$ is the set of tuples

$$\{(\bar{t}, \mathsf{rid}, \mathsf{bid}, \mathsf{tid}, \mathsf{kcnt}) \mid R(\bar{t}) \in D\},$$

where rid identifies R, bid identifies $\mathsf{block}_\Sigma(R(\bar{t}), D_{|R})$, tid identifies $R(\bar{t})$ in $\mathsf{block}_\Sigma(R(\bar{t}), D_{|R})$, and kcnt is $|\mathsf{block}_\Sigma(R(\bar{t}), D)|$. Assume, for every relation R, a unique integer identifier $\#R$. If, for example, R is a binary relation with attributes A and B, and key $\mathsf{key}(R) = \{1\}$, and $D_{|R}$ consists of $R(a_1, b_1)$, $R(a_1, b_2)$, $R(a_1, b_3)$, $R(a_2, c_1)$, $R(a_2, c_2)$, then $Q_R(D)$ is as follows:

A	B	rid	bid	tid	kcnt
a_1	b_1	$\#R$	1	1	3
a_1	b_2	$\#R$	1	2	3
a_1	b_3	$\#R$	1	3	3
a_2	c_1	$\#R$	2	1	2
a_2	c_2	$\#R$	2	2	2

We define Q_R as an SQL query by means of the following view. Let $\bar{\alpha}$, $\bar{\kappa}$, $\bar{\nu}$ denote the list of all attributes, key attributes, and non-key attributes of R, respectively:

```
CREATE VIEW Q_R AS

SELECT ᾱ, #R AS rid,
dense_rank() OVER (ORDER BY κ̄) AS bid,
row_number() OVER (PARTITION BY κ̄ ORDER BY ν̄) AS tid,
count(*) OVER (PARTITION BY κ̄) AS kcnt
FROM R
```

The above SQL view exploits the dense_rank and row_number window functions from the ANSI SQL standard, and thus are supported by all major RDBMSs such as PostgreSQL, MySQL, SQL Server and Oracle. The function row_number ranks (with increasing integers) all the selected rows as specified by the corresponding OVER expression, while dense_rank is similar but two occurrences of the same tuple get the same rank.

Having the above view in place, we can now rewrite the conjunctive query $Q(\bar{x})$ of the form $\exists \bar{y}\,(R_1(\bar{z}_1) \wedge \cdots \wedge R_n(\bar{z}_n))$ into Q^{rew} as follows. Assuming that the SQL version of Q is

$$\text{SELECT } \bar{\alpha} \text{ FROM } R_1, \ldots, R_n \text{ WHERE } \theta,$$

Q^{rew} is the SQL query of arity $|\bar{\alpha}| + 4 \cdot n$ of the form:

```
SELECT ᾱ, R1.rid, R1.bid, R1.tid, R1.kcnt,
       R2.rid, R2.bid, R2.tid, R2.kcnt,
       ...
       Rn.rid, Rn.bid, Rn.tid, Rn.kcnt
FROM QR1 AS R1, QR2 AS R2, ..., QRn AS Rn
WHERE θ ORDER BY ᾱ.
```

Remark. We would like in our experimental analysis to measure the time for constructing $\text{enc}(\text{syn}_{\Sigma,Q}(D))$, and thus our implementation of Algorithm 1 builds it upfront. However, since Q^{rew} orders the output tuples by $\bar{\alpha}$, we could avoid the explicit computation of the whole set $\text{enc}(\text{syn}_{\Sigma,Q}(D))$. Actually, we could iterate over the tuples of $Q^{\text{rew}}(D)$, as the RDBMS computes them, and keep in memory only the (Σ, Q)-synopsis of D for one tuple $\bar{t}$ at a time.

D STATIC QUERY GENERATOR

To generate our stress test queries we exploit a recent query generator [3], which we call *static query generator* (SQG). We call it static since it allows us to tune only static parameters of the query, without taking into account any database.

SQG takes as input a schema $\mathbf{S}$ (the schema of the output CQ), two integers $j \geq 0$ and $c \geq 0$ (the number of joins, and the number of occurrences of constant values, respectively, in the output CQ), a number $0 \leq p \leq 1$ (the percentage of attributes in the output CQ that should be projected), and a function f from $\{R[k] \mid R \in \mathbf{S} \text{ and } k \in [n]\}$, i.e., the set of attributes of the relations of $\mathbf{S}$ ($R[k]$ refers to the k-th attribute of the relation R) to $\mathbf{C}$, which specifies the constant values that can appear in a certain attribute.

The query generator first identifies a set of joinable attribute pairs of the relations of $\mathbf{S}$ by analyzing the foreign-key dependencies of $\mathbf{S}$. It then iteratively creates $j \geq 0$ join conditions by choosing at random an attribute $R[k]$, where $R/n \in \mathbf{S}$ and $k \in [n]$, and then choosing at random an attribute $P[\ell]$, where $P/m \in \mathbf{S}$ and $\ell \in [m]$, that is joinable with $R[k]$. The result of this step is a set of j join conditions of the form $R[k] = P[\ell]$. The query generator then proceeds analogously to generate $c \geq 0$ occurrences of constant values, i.e., constant conditions of the form $R[k] = a$, where $R/n \in \mathbf{S}$, $k \in [n]$, and $a \in f(R[k])$. Clearly, the above randomly generated conditions uniquely determine (up to variable renaming) a set of atoms $A_{j,c}$ over $\mathbf{S}$, that is, the smallest set of atoms over $\mathbf{S}$ such that, for each join condition $R[k] = P[\ell]$, there is an R-atom and a P-atom that mention the same variable at $R[k]$ and $P[\ell]$, respectively, and for each constant condition $R[k] = a$, there is an R-atom that mentions a at $R[k]$. Finally, assuming that T is the set of attributes of the relations occurring in $A_{j,c}$, the query generator chooses at random $\lceil p \cdot |T| \rceil$ attributes of T that should be projected. This gives rise to the CQ $Q_{j,c,p}$ obtained by considering the conjunction of the atoms of $A_{j,c}$ with all the variables occurring at an attribute of T that has been randomly chosen being output variables, and all the other variables being existentially quantified.

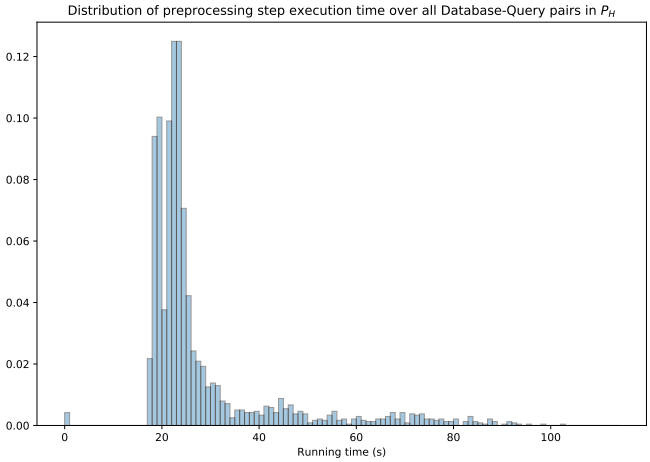


Figure 3: Distribution of preprocessing step running time

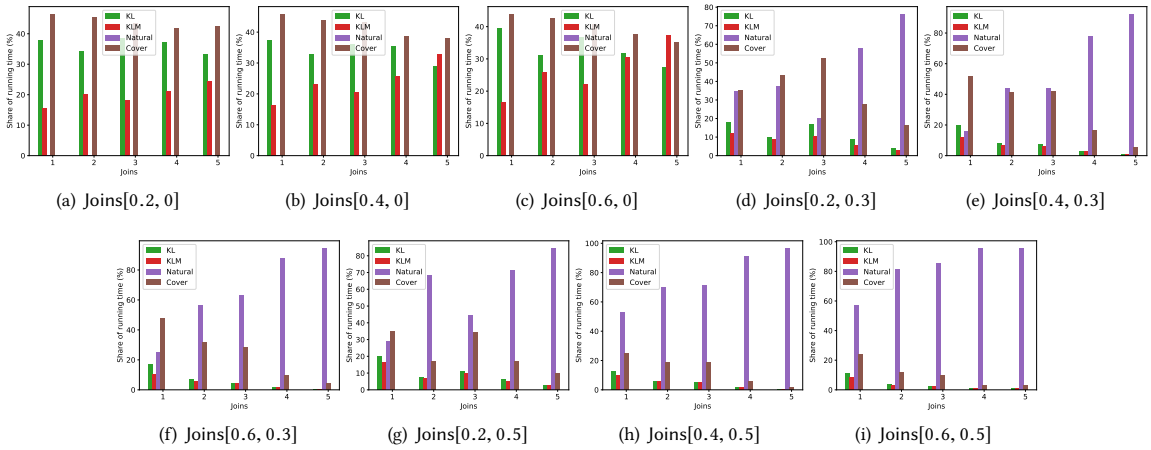
E EXPERIMENTAL EVALUATION

We proceed to give some further details concerning the performance of the preprocessing step over the database-query pairs of P_{H}. We also discuss our experimental analysis based on the join scenarios that is omitted from the main body of the paper. We finally provide a bit more detailed discussion concerning the take-home messages of our analysis.

Preprocessing Step

Our goal here is to illustrate that for the wide variety of database-query pairs (D, Q) that we have generated, $\text{syn}_{\Sigma_{\text{H}},Q}(D)$, which is essentially what we give as an input to the approximation schemes for CQA in our experiments, can be effectively computed via the query rewriting approach from Section 5. Let us stress that our approach is by no means the ultimate way for computing $\text{syn}_{\Sigma_{\text{H}},Q}(D)$ as one may devise a more sophisticated and optimized procedure; this goes beyond the scope of our work. Nevertheless, it turned out that it performs reasonably well in almost all of our scenarios.

In Figure 3, we show the normalized distribution of the running time of our preprocessing step, executed over every pair $(D, Q) \in P_{\text{H}}$, i.e., a bar of height p that corresponds to running time t seconds means that for $p \cdot 100$ percent of pairs (D, Q) in P_{H}, the construction of the set $\text{syn}_{\Sigma,Q}(D)$ via the preprocessing step took between t and $t+1$ seconds. For most pairs (D, Q) of P_{H} it took between 20 and 30 seconds for the computation of the set $\text{syn}_{\Sigma_{\text{H}},Q}(D)$. More precisely, for 80% of the pairs of P_{H}, the preprocessing step completed its execution in less than 30 seconds, for 94% in less than a minute, while the execution time over all pairs never exceeded two minutes. Although there is room for improvement in terms of the runtime



(a) Joins[0.2, 0] (b) Joins[0.4, 0] (c) Joins[0.6, 0] (d) Joins[0.2, 0.3] (e) Joins[0.4, 0.3]

(f) Joins[0.6, 0.3] (g) Joins[0.2, 0.5] (h) Joins[0.4, 0.5] (i) Joins[0.6, 0.5]

Figure 4: Join test scenarios - Joins[*noise*, *balance*]

of the preprocessing step, the take-home message is that relying on such a preprocessing step is not prohibitive in practice.

Join Scenarios

We proceed to investigate how the running time of the approximation schemes for CQA is affected by varying the number of joins occurring in the query. To this end, we consider the family of join scenarios, which consists of 110 test scenarios each consisting of 25 database-query pairs. We ran experiments for all those scenarios but, for the sake of clarity, we present only the results of nine representative scenarios (see Figure 4). Let us stress, however, that the main conclusions that we draw below are in line with what we can conclude from all the 110 join scenarios. Our results regarding all 110 join scenarios (together with our results regarding all the noise and balance scenarios that are omitted from the main body of the paper) can be found below in Section G. When varying the number of joins in the query, each level of joins is associated with a completely different batch of CQs, where the body and the output variables change. Hence, comparing absolute running times in this case makes less sense. Therefore, the plots in Figure 4 show the *share* of the running time of each approximation scheme w.r.t. the running time of the others when focusing on a certain level of joins. In other words, for a join scenario Joins$[p, q]$, we plot, for each $j \in [5]$ and approximation scheme A, the percentage of average time taken by A w.r.t. the other algorithms. As in the analysis of the noise scenarios, we draw different conclusions depending on whether the CQs are Boolean or not.

The Boolean Case. Regardless of the number of joins, Natural is the best performer, taking an extremely small share of the overall running time, and this holds for any level of noise (see plots (a), (b) and (c) in Figure 4). KL improves as the number of joins increases, whereas KLM degrades and eventually performs worse than Cover, for high noise. In any case, Cover is always slower than one of the two KL variants. As already discussed in the analysis of the noise and balance scenarios, Natural is the best performer for any number of joins in the query because $|\mathcal{H}|$ is large, with $(\mathcal{H}, \mathcal{B})$ being the only synopsis in $\mathsf{syn}_{\Sigma_{\mathrm{H}},Q}(D)$, and therefore, $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$ is close to one. On the other hand, the expected values $\mathbb{E}[\mathsf{SampleKL}((\mathcal{H}, \mathcal{B}))]$ and $\mathbb{E}[\mathsf{SampleKLM}((\mathcal{H}, \mathcal{B}))]$, which coincide, are closer to zero. Also, the number of iterations of Cover is linear in $|\mathcal{H}|$.

An interesting observation, which cannot be readily seen from the analysis based on the noise and balance scenarios, is the relationship between KL and KLM. For few joins in the query, KLM performs better than KL (for any level of noise), and the reason is that having the same expected value for their samplers, the variance of $\mathsf{SampleKLM}(\mathcal{H}, \mathcal{B})$ is smaller, and thus the computation of the optimal number of iterations takes much less time. However, we should not forget that SampleKLM is more demanding than SampleKL since SampleKLM needs to iterate over every element of $\mathcal{H}$. This affects the running when we have more joins and higher noise, as $|\mathcal{H}|$ can be quite high, and thus KLM spends most of its time in sampling. Actually, KLM is now worse than KL, and even worse than Cover, as it only depends linearly on $|\mathcal{H}|$.

The Non-Boolean Case. Focusing on non-Boolean CQs (see plots (d), (e) and (f) in Figure 4), Natural is the worst performer, where the difference w.r.t. other algorithms is getting bigger as we increase the number of joins. The KL(M) approximation schemes are now always the best performers, and, as in the Boolean case, KLM is better when we have few joins, but KL catches up as we consider more joins. Cover always performs worse than KL(M).

With higher balance, as we have already discussed, for a given synopsis $(\mathcal{H}, \mathcal{B}) \in \mathsf{syn}_{\Sigma_{\mathrm{H}},Q}(D)$, $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$ is closer to zero. Moreover, by increasing the number of joins, $|\mathsf{syn}_{\Sigma_{\mathrm{H}},Q}(D)|$ increases, and thus, Natural takes even more share of the overall running time. For KL(M), as explained before, high balance means low running time, and as for the Boolean case, KL catches up compared to KLM at large numbers of joins due to the higher computational cost of SampleKLM. For even higher balances (see plots (g), (h) and (i) in Figure 4), the effect of increasing the number of joins on Natural is more evident as $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$ is getting closer to zero.

Take-home Messages

Here is a bit more detailed discussion on the take-home messages of our analysis than the one given in the main body of the paper. They

reveal a striking difference between Boolean and non-Boolean CQs, and highlight the applicability of approximate CQA in practice.

(1) For Boolean CQs, the approximation scheme Natural is the best performer, no matter the amount of noise, and no matter the number of joins in the query, whereas Cover is the worst. Only in the case of CQs with many joins Cover is comparable to KL(M), but in any case, Natural is the way to go.

We remark that the problem of approximating the relative frequency of a tuple is closely related to the problem of approximating the fraction of truth assignments that satisfy a (Block) DNF formula.[6] A database synopsis $(\mathcal{H}, \mathcal{B})$ can be seen as a Block DNF formula, where facts are variables, $\mathcal{H}$ is the set of all clauses, and $\mathcal{B}$ is a partition over the variables. In this setting, sampling from the natural sampling space is generally regarded to be impractical [6, 15] since the fraction we want to approximate can be, in general, very close to zero. However, in the CQA setting, we have seen that in the case of Boolean CQs, relying on the natural sampler is the indicated direction. The reason is that for Boolean CQs, the (only) database synopsis $(\mathcal{H}, \mathcal{B})$ is such that $\mathcal{H}$ is large (as it contains all the homomorphic images of the query), and typically CQs have few joins. The above two properties imply that the relative frequency is actually close to one. Thus, if we cast our problem as the problem of computing the fraction of truth assignments satisfying a Block DNF formula, then this fraction will be close to one.

(2) For non-Boolean CQs, KLM is the way to go in almost all the scenarios, i.e., for any level of noise and for any level of (non-zero) balance of the query. Only for CQs with many joins, and high noise, KL is comparable to KLM. Nevertheless, KL is never going to outperform KLM in practice. The worst algorithms for non-Boolean CQs are Natural and Cover. They perform similarly for low levels of noise, balance and joins, but, in general, Natural is the slowest.

(3) We can safely claim that approximate CQA in the presence of primary key constraints is feasible in practice. We have seen that the preprocessing step, which is responsible for computing the synopses, has completed its execution in less than 30 seconds in most cases. Furthermore, for modest scenarios, which is what we expect to face in practice, the running time of the best performing approximation scheme is reasonable. For example, for 50% noise, 50% balance, and CQs with 3 joins, the runtime of the best performer is at most 6 seconds, and this decreases to 3 seconds for 30% noise, and 30% balance. Hence, after executing the preprocessing step, one can choose the appropriate approximation scheme for the database and query at hand, according to the conclusions of items (1) and (2), leading to an overall process whose running time is quite encouraging considering the hardness of the problem in question. Thus, we strongly believe that finding approximate solutions to CQA is feasible in practice.

At this point, let us stress again that our approach, described in Section C, for computing the synopses is by no means the ultimate one, and there is room for improvement. Moreover, the performance of the approximation schemes for CQA can greatly benefit from a parallel implementation of the sampling phase without additional synchronization overhead.

[6] A Block DNF formula is a positive DNF formula, where its variables are partitioned into $X_1, \ldots, X_n$, and only truth assignments that make exactly one variable from each X_i true are considered.

F VALIDATING OUR RESULTS

Recall that we concluded Section 7.2 by noticing that we have also experimentally validated our main conclusions concerning the main approximation schemes for CQA (see take-home messages (1) and (2) above). We proceed to actually perform this validation. To this end, we rely on a different batch of test scenarios, which we call validation scenarios, that are closer to real-world use cases.

Validation Scenarios

For our validation scenarios we rely on a subset of the query workloads provided by two TPC benchmarks: TPC-H once again, and TPC-DS. The queries coming with the above two benchmarks have been manually designed to resemble a typical query workload of a data warehouse setting. The TPC-DS benchmark also comes with its own primary keys, but it has a different structure than TPC-H; it is actually a combination of multiple snowflake schemas, and thus, it contains more relations (24) and columns (up to 34).

Generating the Data. Similarly to what we did for the stress test scenarios, we considered the TPC-H and TPC-DS schemas $\mathbf{S}_{\mathrm{H}}$ and $\mathbf{S}_{\mathrm{DS}}$, together with their set of primary keys Σ_{H} and Σ_{DS}, respectively. We generated the consistent databases D_{H} and D_{DS} using the data generation tools provided by the TPC-H and TPC-DS benchmarks, respectively, with scale factor 1GB. The databases D_{H} and D_{DS} contain roughly 9 and 20 million tuples, respectively.

Selecting the Queries. The TPC-H and TPC-DS benchmarks provide a manually-curated workload of query templates that can be instantiated using the provided query generation tools. Such queries are meant to represent typical queries in a data warehousing scenario, and in each workload each query template has a unique integer identifier. Since some of the templates do not correspond to conjunctive queries (they use some form of negation), we generated concrete instantiations of a selection of 17 positive queries from the above benchmarks, after removing aggregate functions in the SELECT clause. We report below the queries we considered; Q^i_{B} denotes the instantiation we obtained from the i-th query template of the query workload of benchmark TPC-B, with $\mathrm{B} \in \{\mathrm{H}, \mathrm{DS}\}$:

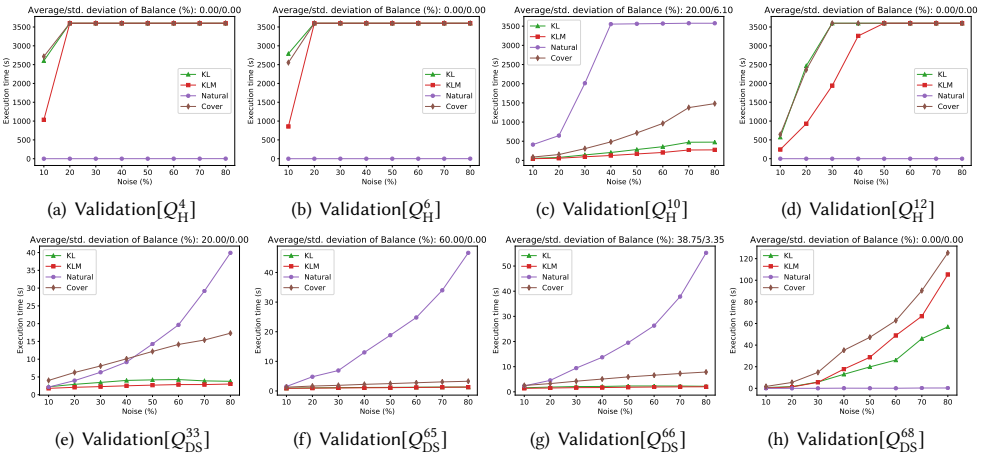
$$\mathcal{Q}_{\mathrm{H}} = \{Q^1_{\mathrm{H}}, Q^4_{\mathrm{H}}, Q^5_{\mathrm{H}}, Q^6_{\mathrm{H}}, Q^8_{\mathrm{H}}, Q^{10}_{\mathrm{H}}, Q^{12}_{\mathrm{H}}, Q^{14}_{\mathrm{H}}, Q^{19}_{\mathrm{H}}\}$$

and

$$\mathcal{Q}_{\mathrm{DS}} = \{Q^1_{\mathrm{DS}}, Q^{33}_{\mathrm{DS}}, Q^{60}_{\mathrm{DS}}, Q^{62}_{\mathrm{DS}}, Q^{65}_{\mathrm{DS}}, Q^{66}_{\mathrm{DS}}, Q^{68}_{\mathrm{DS}}, Q^{82}_{\mathrm{DS}}\}.$$

Devising our Validation Scenarios. Since our intention is to assess our results over scenarios that are closer to real settings, our batch of queries is coming from the above workloads. Hence, we have no control on the balance and the number of joins of each CQ, but we can freely vary the noise. We consider databases with noise between 10% and 80%. Let us note that we do not go beyond 80% for a couple of reasons: (i) for some of the considered queries, the process of generating noise using our query-aware noise generator is becoming extremely demanding without terminating in a reasonable amount of time, and (ii) databases with noise beyond 80% are unlikely to appear in a real-life scenario.

For each $\mathrm{B} \in \{\mathrm{H}, \mathrm{DS}\}$, and for each CQ $Q \in \mathcal{Q}_{\mathrm{B}}$, we considered the *validation scenario of* Q, dubbed Validation$[Q]$, consisting of 8 databases $D_1, \ldots, D_8$ that are inconsistent w.r.t. Σ_{B}. For each $i \in [8]$, D_i was generated by calling our query-aware noise generator with



(a) Validation$[Q_{\mathrm{H}}^{4}]$ (b) Validation$[Q_{\mathrm{H}}^{6}]$ (c) Validation$[Q_{\mathrm{H}}^{10}]$ (d) Validation$[Q_{\mathrm{H}}^{12}]$

(e) Validation$[Q_{\mathrm{DS}}^{33}]$ (f) Validation$[Q_{\mathrm{DS}}^{65}]$ (g) Validation$[Q_{\mathrm{DS}}^{66}]$ (h) Validation$[Q_{\mathrm{DS}}^{68}]$

Figure 5: Validation scenarios

input the (consistent database) D_{B}, the set of primary keys Σ_{B}, the query Q, noise level $i/10$, and minimum and maximum block size equal to 2 and 5, respectively. Summing up, we considered 17 validation scenarios (9 based on TPC-H, and 8 based TPC-DS) each consisting of 8 inconsistent databases of increasing noise.

Experimental Validation Analysis

Although we ran experiments for all the validation scenarios, for the sake of clarity, we present only the results of eight representative scenarios (see Figure 5). Let us stress that all the validation scenarios provide results that indeed confirm our main outcomes regarding the approximation schemes for CQA. Our results for all 17 validation scenarios are presented below in Section H.

The plots depicted in Figure 5 show how the running time of the approximation schemes is affected by varying the noise focusing on the CQs Q_{H}^{i}, for $i \in \{4, 6, 10, 12\}$, and Q_{DS}^{i}, for $i \in \{33, 65, 66, 68\}$. Each plot also specifies the average and the standard deviation of the balance of the corresponding query Q over the inconsistent databases from Validation$[Q]$ up to 4 decimal digits.

Observe that for most of the CQs presented in Figure 5, the standard deviation of their balance is very close to zero (it is zero up to 4 decimal digits), and hence, it does not change much for different levels of noise. The only exceptions are the CQs Q_{H}^{10} and Q_{DS}^{65} considered in plots (c) and (g) of Figure 5, respectively. However, even in those cases the standard deviation is low, which implies that the balance of such queries does not deviate too much, on average, when varying the noise. Therefore, we can safely compare the plots in Figure 5 with the corresponding ones in Figure 1 about the noise scenarios where the balance and the number of joins are fixed.

Validating Take-home Message (1). The queries Q_{H}^{4}, Q_{H}^{6}, Q_{H}^{12} and Q_{DS}^{68} have an average balance of zero over all eight inconsistent databases on which they have been evaluated. Actually, except for Q_{H}^{6}, which is a Boolean query, the other queries are non-Boolean but behave like Boolean as the number of output tuples is very small when compared to the number of homomorphic images, which in turn implies that the balance is close to zero. For Q_{H}^{4}, the only attribute in the SELECT clause is a categorical one, which takes very few values in the database; the same applies to Q_{H}^{6}. Regarding Q_{DS}^{68}, the low balance is because, although the number of homomorphic images is large, and the output attributes are not categorical, due to the WHERE clause, they take only a few distinct values. This causes the number of output tuples to stay low w.r.t. the number of homomorphic images, and hence, the balance is low on average.

The above discussion essentially tells us that we should compare the trend of the runtime of the approximation schemes according to plots (a), (b), (d) and (h) of Figure 5 with the trend of the runtime according to the plots (a), (b) and (c) of Figure 1, which consider CQs of zero balance. It is evident that the overall trend is confirmed as the approximation scheme Natural is the best performer for any level of noise, whereas all the other algorithms quickly time out for most CQs. This actually validates our first take-home message concerning Boolean CQs, and CQs of low balance.

Validating Take-home Message (2). The average balance of Q_{H}^{10} and Q_{DS}^{33} is 0.20, while that of Q_{DS}^{66} is 0.3575. Therefore, we can compare the trend of the running time of the algorithms according to the plots of Figure 5 with the trend of the runtime according to plots (d), (e) and (f) of Figure 1. We can see that also in this case the overall trend is confirmed with KLM being the fastest approximation scheme, regardless of the noise, whereas Natural is the slowest where its running time quickly increases as the noise increases. Finally, the average balance for query Q_{DS}^{65} is 0.60, and comparing the trend of the running time according to plot (f) of Figure 5 with that according to plots (g), (h) and (i) of Figure 1, which consider CQs of high balance, we can also confirm our second take-home message concerning non-Boolean CQs of high balance.

G ALL TEST SCENARIOS

For the sake of completeness, we provide the plots for all the test scenarios. Figure 6 collects the plots for all Noise scenarios with balance up to 0.4, while Figure 7 collects the plots for all Noise scenarios with balance between 0.5 and 1.0. Figure 8 collects the

plots for all Balance scenarios with noise up to 0.5, while Figure 9 collects the plots for all Balance scenarios with noise between 0.6 and 1.0. Finally, Figure 10 collects the plots for all Join scenarios with balance up to 0.2, Figure 11 with balance between 0.3 and 0.5, Figure 12 with balance between 0.6 and 0.8, and Figure 13 with balance between 0.9 and 1.0.

H ALL VALIDATION SCENARIOS

Finally, again for the sake of completeness, we present the plots for all the validation scenarios, including those not shown in Section F. Figure 14 collects the plots for the TPC-H validation scenarios, while Figure 15 the plots for the TPC-DS validation scenario. Recall that Q^i_{B} denotes an instantiation of the i-th query template of the query workload of benchmark TPC-B, with $\mathrm{B} \in \{\mathrm{H}, \mathrm{DS}\}$.

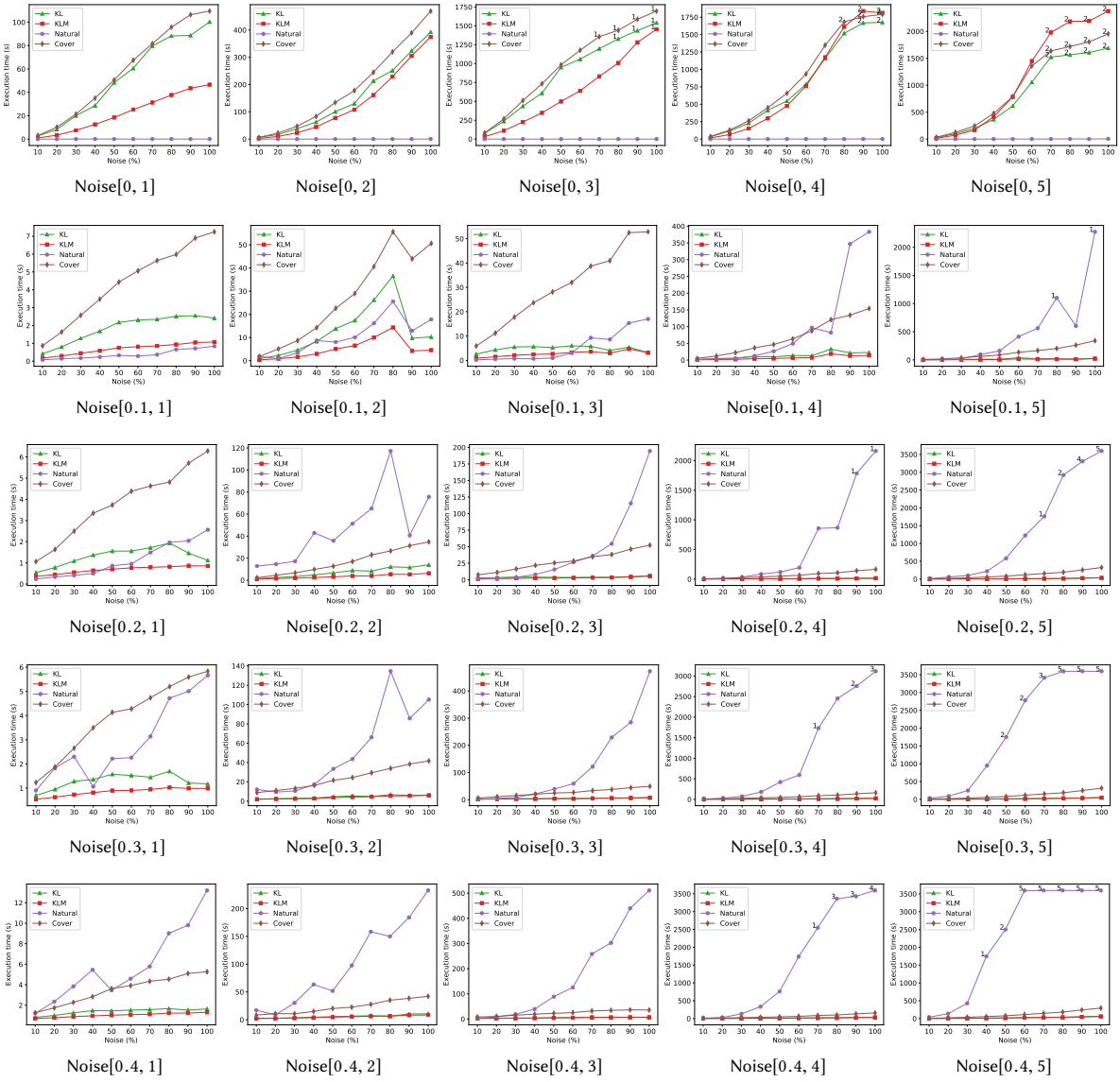


Figure 6: Noise scenarios Noise[*balance*, *joins*] **with balance up to** 0.4

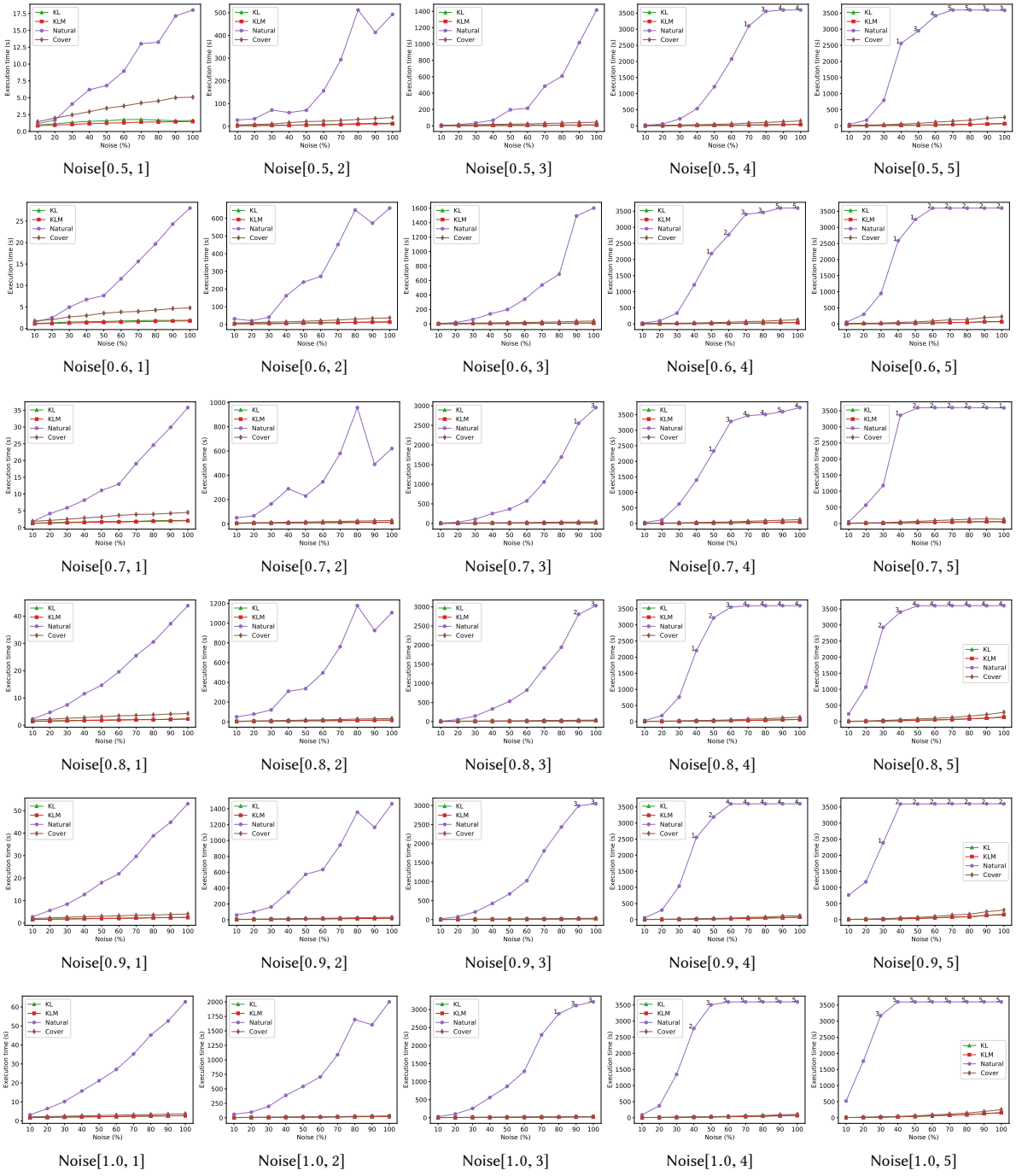


Figure 7: Noise scenarios Noise[*balance*, *joins*] **with balance between** 0.5 **and** 1.0

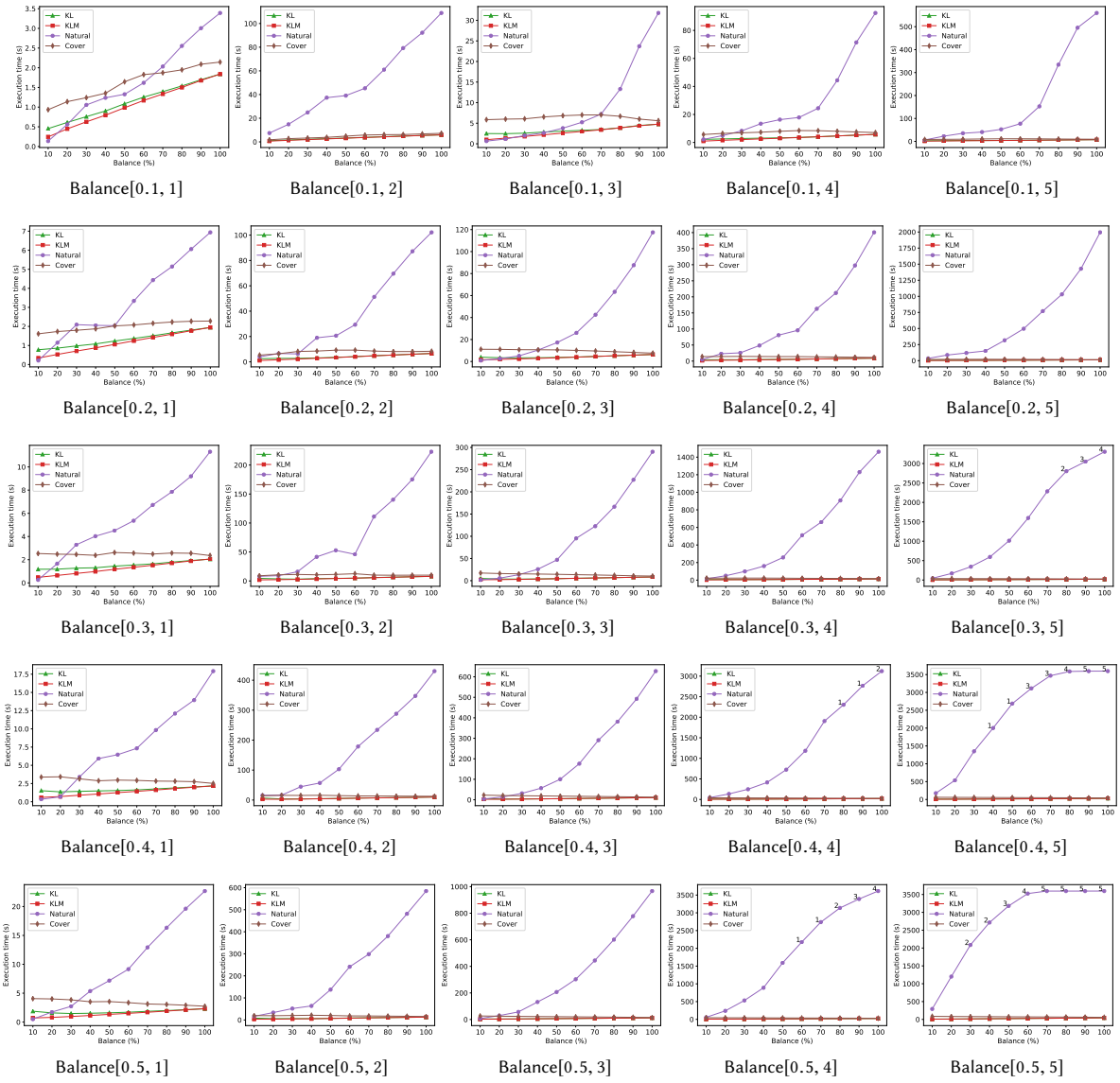


Figure 8: All Balance scenarios Balance[*noise*, *joins*] **with noise up to** 0.5

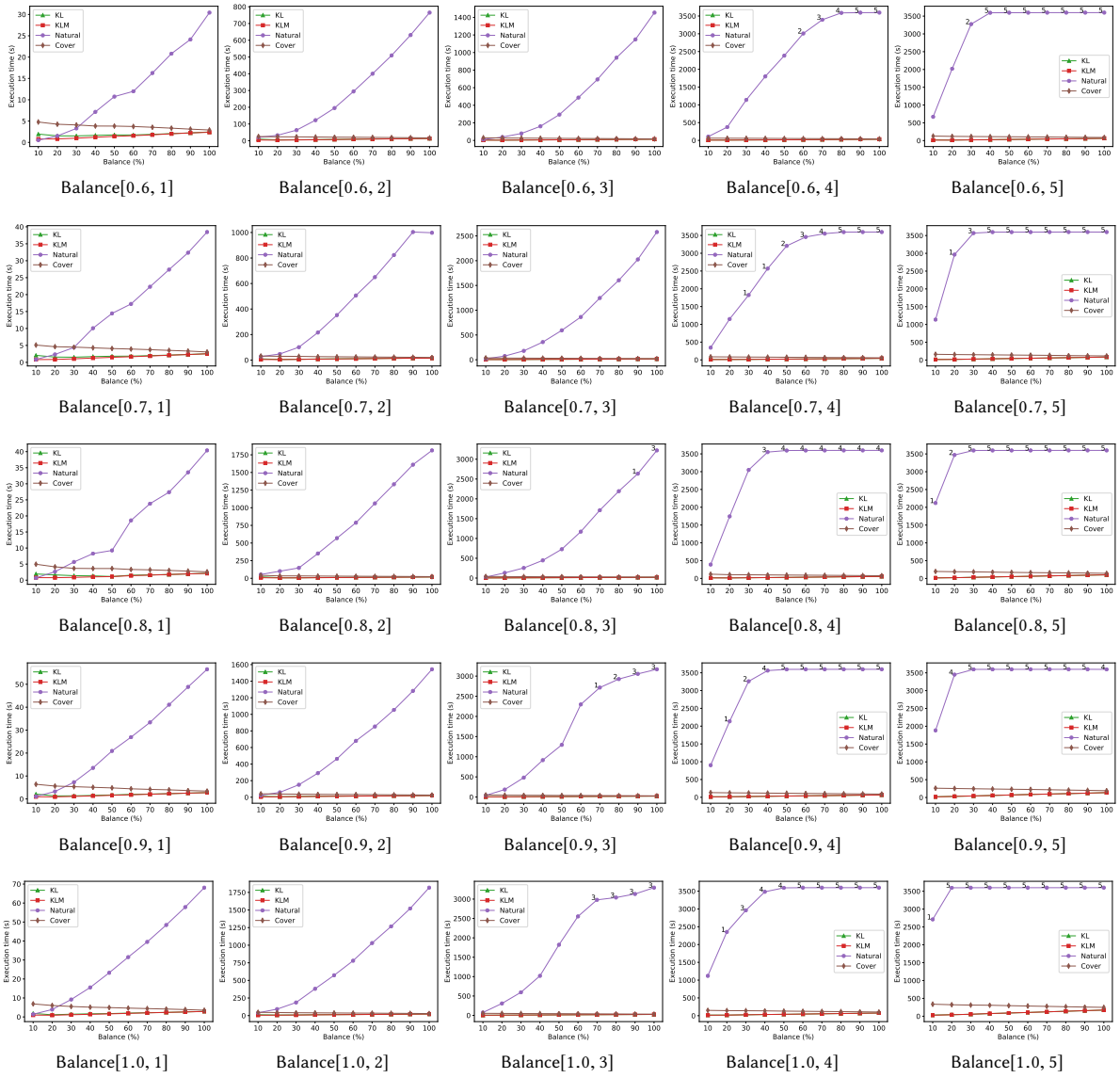


Figure 9: Balance scenarios Balance[*noise*, *joins*] **with noise between** 0.6 **and** 1.0

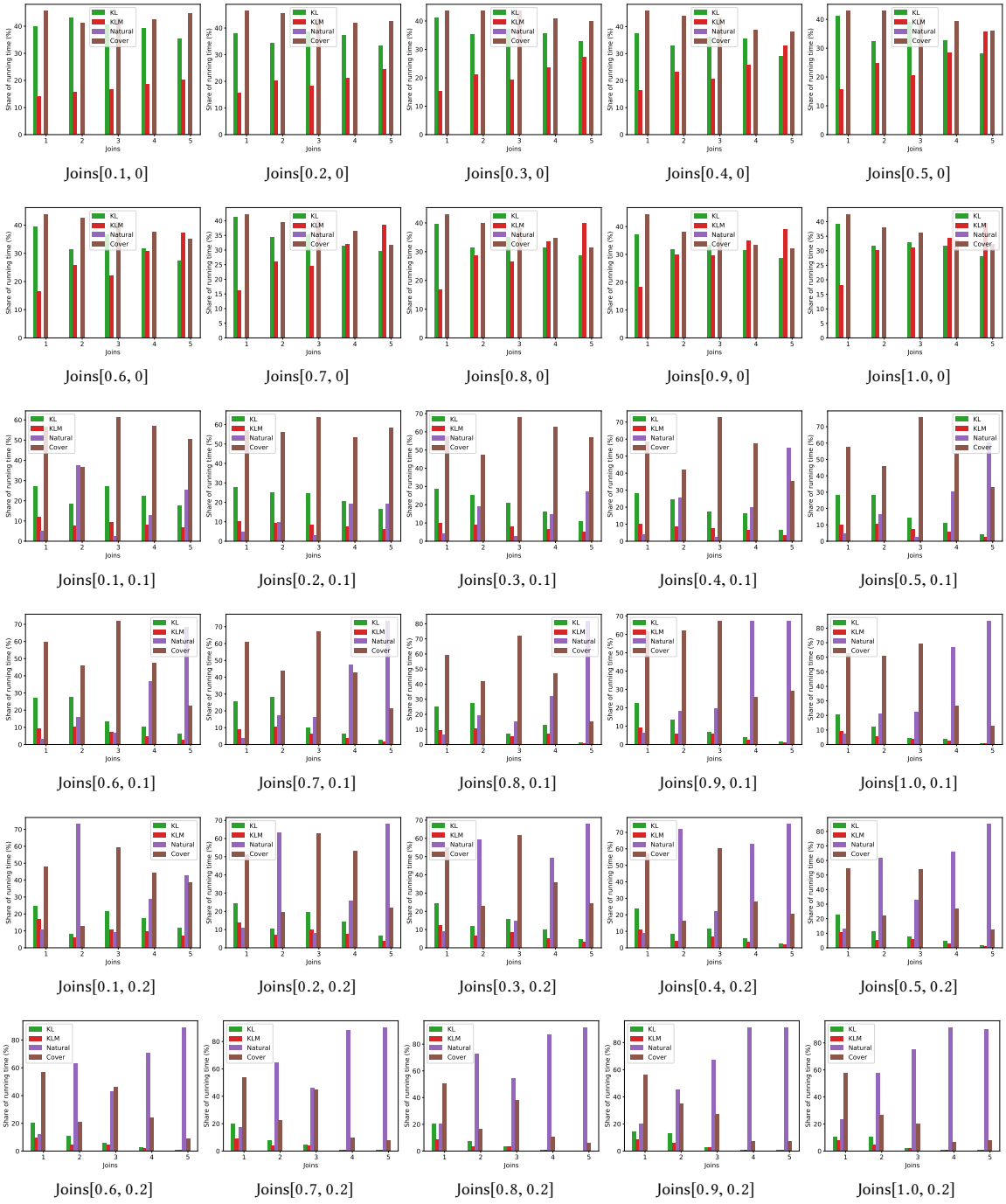


Figure 10: Join scenarios Join[*noise*, *balance*] **with balance up to** 0.2

Figure 11: Join scenarios Join[*noise*, *balance*] **with balance between** 0.3 **and** 0.5

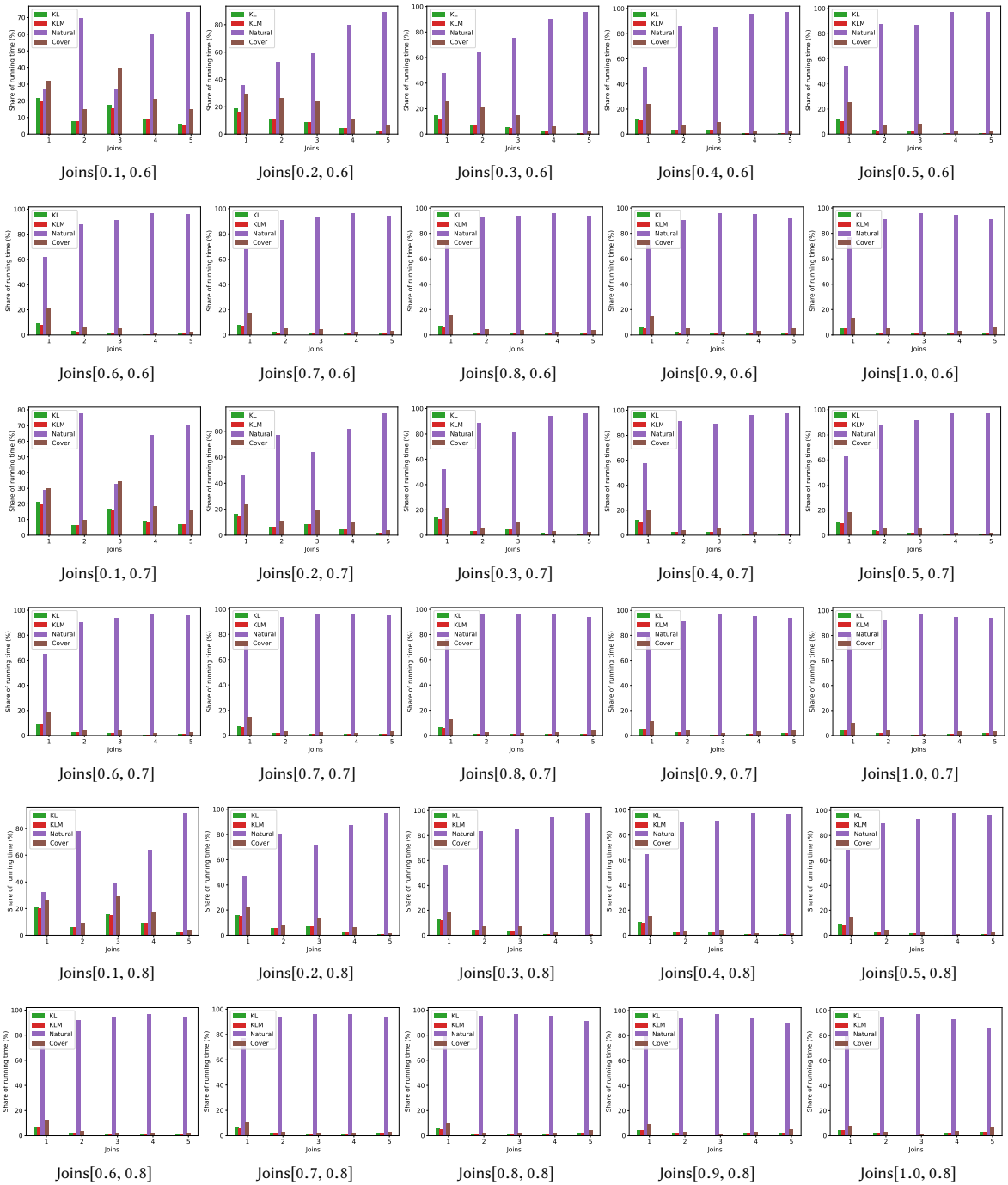


Figure 12: Join scenarios Join[*noise*, *balance*] **with balance between** 0.6 **and** 0.8

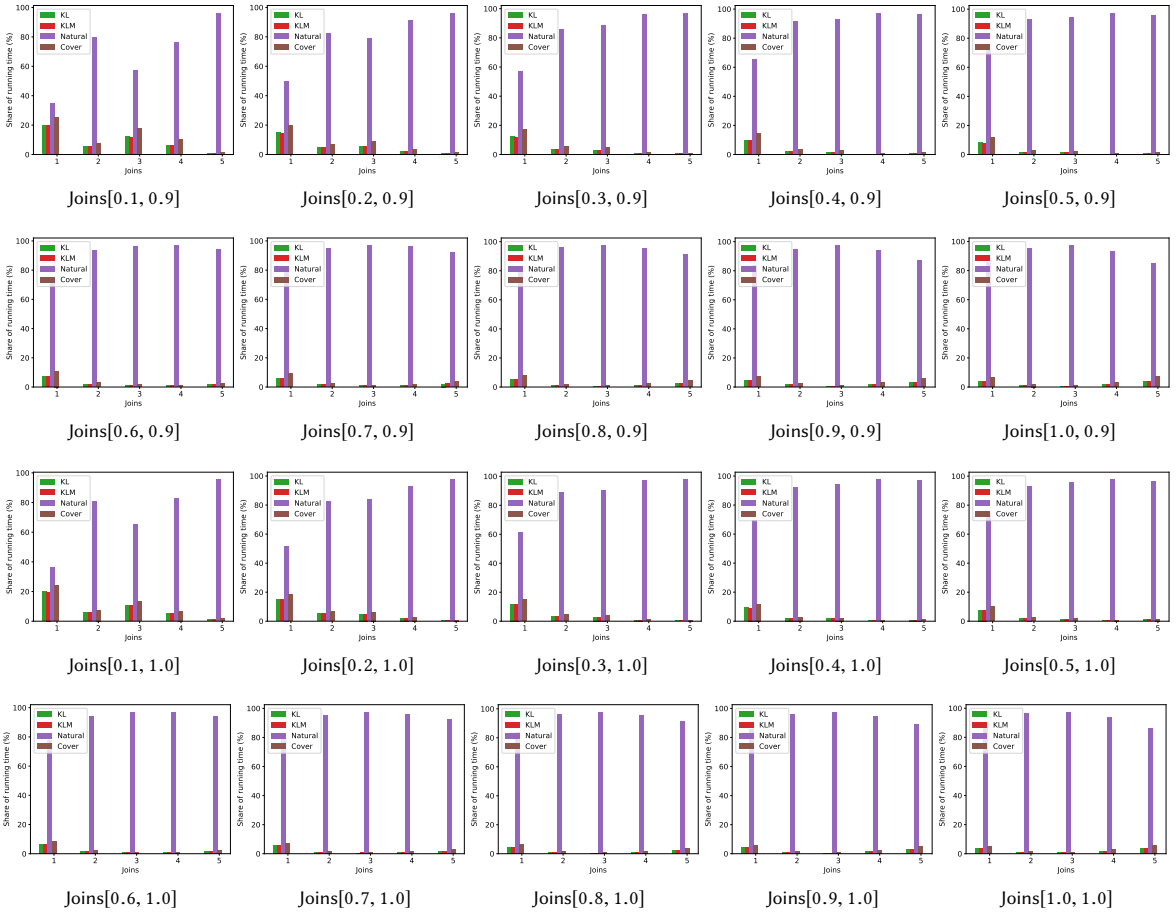


Figure 13: Join scenarios Join[*noise*, *balance*] **with balance between** 0.9 **and** 1.0

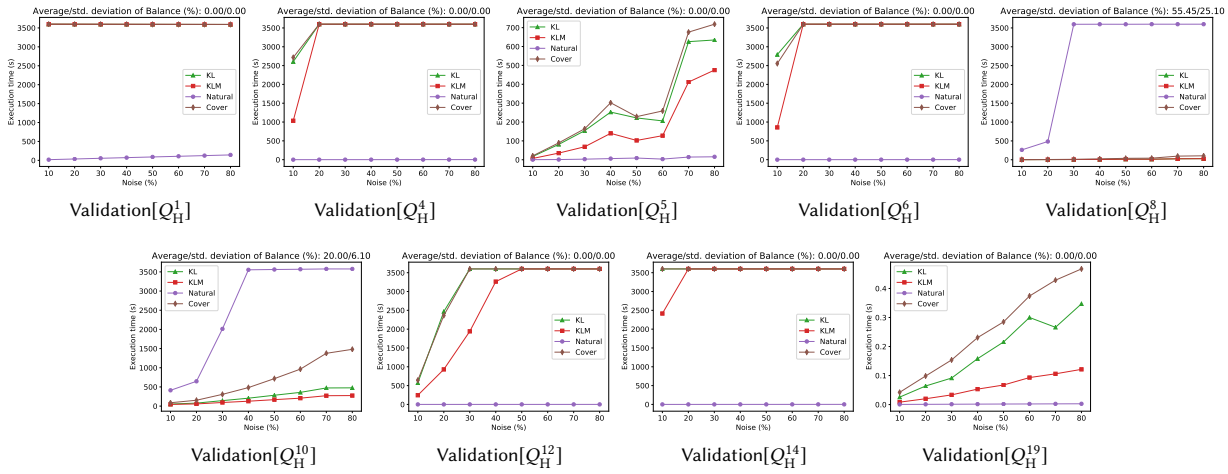


Figure 14: Validation scenarios based on TPC-H

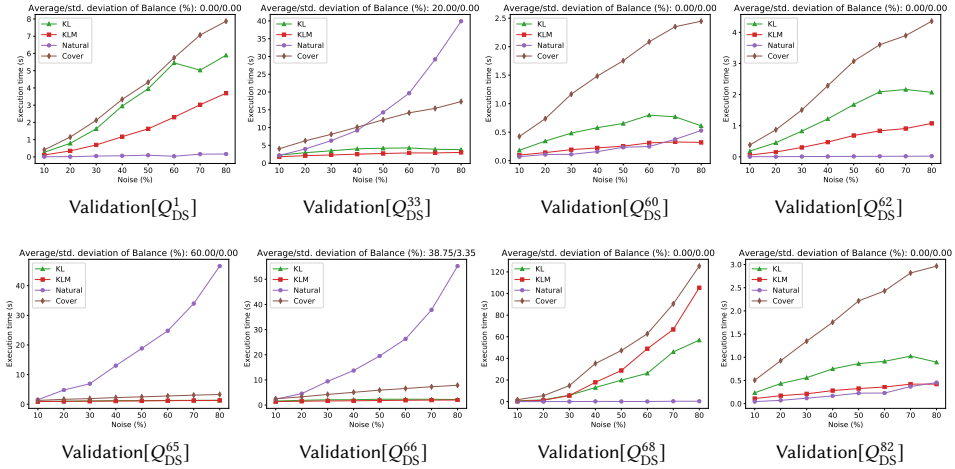


Figure 15: Validation scenarios based on TPC-DS