# Benchmarking Approximate Consistent Query Answering

(Discussion Paper)

Marco Calautti[1], Marco Console[2] and Andreas Pieris[3]

[1]DISI, University of Trento, Italy
[2]Sapienza, University of Rome, Italy
[3]University of Edinburgh, 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. A more informative notion is the percentage of repairs in which a candidate answer is true, called relative frequency. Computing this percentage is intractable in general, but for the relevant setting of conjunctive queries and primary keys, data-efficient randomized approximation schemes exist. Our goal is to perform a thorough experimental evaluation and comparison of those approximation schemes and provide new insights on which technique is indicated depending on key characteristics of the input.

## 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 that differs somehow minimally from the original database $D$, 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 [2]:

**Example 1.** *Consider the single relation* Employee(id, name, dept) *where the attribute* id *is the key of the relation* Employee. *Consider also the database $D$ 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*

*SEBD 2021: The 29th Italian Symposium on Advanced Database Systems, September 5-9, 2021, Pizzo Calabro (VV), Italy*
marco.calautti@unitn (M. Calautti); console@diag.uniroma1.it (M. Console); apieris@inf.ed.ac.uk (A. Pieris)

2 *work in the same department. This query is true only in two repairs, thus, according to certain answers, is not entailed.* ■

CQA has been extensively studied both in theory (see, e.g., [3, 4, 5]), and in practice (see, e.g., [6, 7]). 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 frequently observed (e.g., see [2, 8, 9]), the former is too strict, while the latter is not very useful in a practical context.

**A Refined Approach.** A simple step in obtaining more information for a candidate answer is to consider its relative frequency, i.e., the percentage of repairs that entail the answer [2]. In Example 1, the relative frequency of the empty tuple (i.e., the only candidate answer for Boolean queries) is 50% since, out of four repairs in total, only two satisfy the query. However, computing this simple measure is #P-hard, even in data complexity, i.e., when the query and the constraints are fixed, and even if we focus on conjunctive queries and primary keys [10]. Hence, the way to proceed is to give up exact solutions, and target approximations.

**Data-efficient Approximations.** Approximation algorithms have been crucial for tackling intractable problems in different areas of Data Management. Examples are in the context of approximating certain answers over incomplete databases, where different approximation algorithms have been devised and experimentally evaluated [11, 12, 13]. In the case of inconsistent databases, data-efficient randomized algorithms exist, in the relevant setting of conjunctive queries and primary keys, that approximate the relative frequency of a candidate answer within a desired error, with high probability [2]. These rely on techniques that have been originally proposed to approximate the number of satisfying assignments of DNF Boolean formulas [14]. However, no corresponding infrastructure for experimentally evaluating such techniques exists.

**Main Objective.** The main objective of this work is to provide a benchmark for these randomized approximation schemes, when considering primary keys and conjunctive queries, covering a wide range of scenarios, and exploit this benchmark to asses how the performance of the approximation schemes is affected by key parameters of the input.[1]

## 2. Preliminaries

**Relational Databases.** A *schema* $\mathbf{S}$ is a finite set of relation symbols (or predicates) with associated arity. A *database* $D$ over a schema $\mathbf{S}$ is a set of facts of the form $R(\bar{t})$, where $R \in \mathbf{S}$, and $\bar{t} = t_1, \ldots, t_n$ is a tuple of terms that are drawn from a countably infinite set $\mathbf{C}$ of constants. We write $\mathsf{dom}(D)$ for the *active domain* of $D$, that is, the set of constants occurring in $D$.

**Primary Key Constraints.** A *key constraint* (or *key*) $\kappa$ over a schema $\mathbf{S}$ is an expression $\mathsf{key}(R) = \{1, \ldots, m\}$, where $R \in \mathbf{S}$ has arity $n$, and $m \leq n$; we also call it an $R$-key. A database $D$ satisfies $\kappa$ if, for every $R(\bar{t}), R(\bar{s}) \in D$, $\bar{t}[i] = \bar{s}[i]$ for each $i \in \{1, \ldots, m\}$ implies $\bar{t} = \bar{s}$. We say that $D$ is *consistent* w.r.t. a set $\Sigma$ of keys, written $D \models \Sigma$, if $D$ satisfies each key in $\Sigma$; otherwise, it is *inconsistent.* A set of *primary keys* $\Sigma$ is a set of keys such that, for each predicate $R$, $\Sigma$ has at most one $R$-key. For $\alpha = R(c_1, \ldots, c_n)$, the *key value* of

[1]An extended version of this paper has been accepted to PODS 2021, and can be found at https://tinyurl.com/f8sev5pj. Source code and test scenarios can be found at https://gitlab.com/mcalautti/cqabench.

$\alpha$ w.r.t. $\Sigma$, denoted $\mathsf{key}_\Sigma(\alpha)$, is defined as $\langle R, \langle c_1, \ldots, c_m \rangle\rangle$, if $\mathsf{key}(R) = \{1, \ldots, m\} \in \Sigma$, and $\langle R, \langle c_1, \ldots, c_n \rangle\rangle$ otherwise. Let $\mathsf{block}_\Sigma(\alpha, D) = \{\beta \in D \mid \mathsf{key}_\Sigma(\beta) = \mathsf{key}_\Sigma(\alpha)\}$, and $\mathsf{block}_\Sigma(D) = \{\mathsf{block}_\Sigma(\alpha, D) \mid \alpha \in D\}$. A *repair* of $D$ w.r.t. $\Sigma$ is a database obtained by choosing one fact from each $B \in \mathsf{block}_\Sigma(D)$. We denote the set of repairs of $D$ w.r.t $\Sigma$ as $\mathsf{rep}(D, \Sigma)$.

**Conjunctive Queries.** A *conjunctive query* $Q(\bar{x})$ over a schema $\mathbf{S}$ is a first-order expression of the form $\exists \bar{y} \left(R_1(\bar{z}_1) \wedge \cdots \wedge R_n(\bar{z}_n)\right)$ that mentions only atoms over $\mathbf{S}$, and has free variables $\bar{x}$. For a tuple $\bar{t} \in \mathbf{C}^{|\bar{x}|}$, $Q(\bar{t})$ denotes $Q$ after replacing the $i$-th variable in $\bar{x}$ with the $i$-th constant in $\bar{t}$. A *homomorphism* from $Q$ 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 $i \in [n]$. We use $Q \xrightarrow{h} D$ to say that $h$ is a homomorphism from $Q$ to $D$. The answer to $Q(\bar{x})$ over a database $D$, denoted $Q(D)$, is the set of tuples $\{h(\bar{x}) \in \mathsf{dom}(D)^{|\bar{x}|} \mid Q(\bar{x}) \xrightarrow{h} D\}$.

**Consistent Query Answering.** Consider a database $D$, a set $\Sigma$ 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. $D$, $\Sigma$ and $Q$ is the ratio $\mathsf{R}_{D,\Sigma,Q}(\bar{t}) = |\{D' \in \mathsf{rep}(D, \Sigma) \mid \bar{t} \in Q(D')\}|/|\mathsf{rep}(D, \Sigma)|$. Our main problem is:

| | |
|---|---|
| PROBLEM : | $\mathsf{CQA}$ |
| INPUT : | A database $D$, a set $\Sigma$ of primary keys, and a CQ $Q(\bar{x})$. |
| OUTPUT : | The set $\left\{(\bar{t}, \mathsf{R}_{D,\Sigma,Q}(\bar{t})) \mid \bar{t} \in \mathsf{dom}(D)^{|\bar{x}|} \text{ and } \mathsf{R}_{D,\Sigma,Q}(\bar{t}) > 0\right\}$. |

That is, output, for each candidate answer tuple $\bar{t}$, its relative frequency, when this is not zero. We dub the problem of computing $\mathsf{R}_{D,\Sigma,Q}(\bar{t})$, given $D, \Sigma, Q, \bar{t}$, $\mathsf{RelativeFreq}$. We know that $\mathsf{RelativeFreq}$ is #P-hard, in data complexity. Clearly, having an efficient way of approximating the relative frequency of a tuple will immediately provide an algorithm for approximately solving $\mathsf{CQA}$. Hence, in the rest, we focus on $\mathsf{RelativeFreq}$.

**Approximate CQA.** A *(data-efficient) randomized approximation scheme* for the problem $\mathsf{RelativeFreq}$ is a randomized algorithm $\mathsf{A}$ that takes a database $D$, a set $\Sigma$ 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 $||D|| + ||\bar{t}||$,[2] $1/\epsilon$ and $\log(1/\delta)$, and produces a random variable $\mathcal{A}$ such that $\mathsf{Pr}\left(|\mathcal{A} - \mathsf{R}_{D,\Sigma,Q}(\bar{t})| \leq \epsilon \cdot \mathsf{R}_{D,\Sigma,Q}(\bar{t})\right) \geq 1 - \delta$.

## 3. Approximation Schemes

In principle, an approximation scheme for $\mathsf{RelativeFreq}$ would require to access the input database $D$. However, doing this naively is prohibitive in practice since, in general, $D$ is very large. However, an approximation scheme for $\mathsf{RelativeFreq}$ only needs a small part of the database, which we call synopsis. In what follows, let $D, \Sigma, Q(\bar{x}), \bar{t}$ be a database, a set of primary keys, a CQ and a tuple in $\mathsf{dom}(D)^{|\bar{x}|}$, respectively.

The $(\Sigma, Q)$*-synopsis of* $D$ *for* $\bar{t}$ is the pair $(\mathcal{H}, \mathcal{B})$, where $\mathcal{H} = \{h(Q) \mid Q(\bar{t}) \xrightarrow{h} D, \text{ and } h(Q) \models \Sigma\}$ and $\mathcal{B} = \{\mathsf{block}_\Sigma(\alpha, D) \mid \alpha \in \cup_{H \in \mathcal{H}} H\}$. That is, the $(\Sigma, Q)$-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

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

set $\mathcal{B}$). We also let $\mathsf{db}(\mathcal{B}) = \{\{\alpha_1, \dots, \alpha_n\} \mid \langle\alpha_1, \dots, \alpha_n\rangle \in \times_{B \in \mathcal{B}} B\}$, i.e., the set of databases that can be formed by keeping exactly one atom from each member $B$ of $\mathcal{B}$. Finally, $\mathsf{R}_{(\mathcal{H},\mathcal{B})} = |\{I \in \mathsf{db}(\mathcal{B}) \mid H \subseteq I \text{ for some } H \in \mathcal{H}\}||\mathsf{db}(\mathcal{B})|$.[3]

We show that the $(\Sigma, Q)$-synopsis of a database $D$ for a tuple $\bar{t}$ can be efficiently constructed, and it contains enough information that allows us to compute the relative frequency of $\bar{t}$.

**Lemma 2.** *If $(\mathcal{H}, \mathcal{B})$ is the $(\Sigma, Q)$-synopsis of $D$ for $\bar{t}$, then :*1) *$(\mathcal{H}, \mathcal{B})$ can be computed in polynomial time in $||D|| + ||\bar{t}||$;* 2) $\mathsf{R}_{D,\Sigma,Q}(\bar{t}) = \mathsf{R}_{(\mathcal{H},\mathcal{B})}$; 3) $\mathsf{R}_{D,\Sigma,Q}(\bar{t}) = 0$ *iff* $\mathcal{H} = \emptyset$.

## 3.1. Monte Carlo Approach

Let $(\mathcal{H}, \mathcal{B})$ be the $(D, \Sigma)$-synopsis of some tuple $\bar{t}$ and let $\mathsf{Sample}$ be a randomized procedure that with input $(\mathcal{H}, \mathcal{B})$, computes a random number in $[0, 1]$. Being randomized, its output is determined by the underlying sampling space obtained from $(\mathcal{H}, \mathcal{B})$. $\mathsf{Sample}(\mathcal{H}, \mathcal{B})$ produces a random variable, and a crucial problem for us is computing its expected value $\mathbb{E}[\mathsf{Sample}(\mathcal{H}, \mathcal{B})]$.

$\mathbb{E}[\mathsf{Sample}(\mathcal{H}, \mathcal{B})]$ can be approximated by performing the following: (1) $S := 0$, (2) for $N$ times do the following: $S := S + \mathsf{Sample}(\mathcal{H}, \mathcal{B})$, 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 $\mathbb{E}[\mathsf{Sample}(\mathcal{H}, \mathcal{B})]$. From [15], we know that we can compute the *minimum* number (up to constant factors) of iterations $N$ such that the above algorithm approximates $\mathbb{E}[\mathsf{Sample}(\mathcal{H}, \mathcal{B})]$ within a given error $\epsilon$, and with probability at least $1 - \delta$, for some given $0 < \delta < 1$. By exploiting this approach in the algorithm discussed above, we obtain the algorithm $\mathsf{MonteCarlo}[\mathsf{Sample}]$, which is parametrized with a randomized procedure $\mathsf{Sample}$. Let $||\mathcal{H}, \mathcal{B}|| = |\mathcal{H}| + \max_{H \in \mathcal{H}}\{||H||\} + ||\mathcal{B}||$. From [15], we know that:

(*) *If* $\mathsf{Sample}$ *runs in polynomial time in* $||\mathcal{H}, \mathcal{B}||$ *and* $\mathbb{E}[\mathsf{Sample}(\mathcal{H}, \mathcal{B})] > 1/\mathsf{pol}(||\mathcal{H}, \mathcal{B}||)$*, when* $\mathbb{E}[\mathsf{Sample}(\mathcal{H}, \mathcal{B})] > 0$*, for some polynomial* $\mathsf{pol}$*, then* $\mathsf{MonteCarlo}[\mathsf{Sample}]$ *is a data-efficient randomized approximation scheme for computing* $\mathbb{E}[\mathsf{Sample}(\mathcal{H}, \mathcal{B})]$.[4]

By combining the above property and Lemma 2, an approximation scheme for $\mathsf{RelativeFreq}$ can be obtained by devising the right sampling procedure $\mathsf{Sample}$.

**Natural Sampling Space.** The first sampler, which we call $\mathsf{SampleNatural}$, takes as input $(\mathcal{H}, \mathcal{B})$, and has $\mathbb{E}[\mathsf{SampleNatural}(\mathcal{H}, \mathcal{B})] = \mathsf{R}_{(\mathcal{H},\mathcal{B})}$, i.e., its expected value precisely coincides with the relative frequency. This sampler simply performs the following: *1)* sample a database $I \in \mathsf{db}(\mathcal{B})$ uniformly at random (u.a.r.); *2)* if $H \subseteq I$, for some $H \in \mathcal{H}$, output 1, otherwise output 0. We can prove that $\mathsf{SampleNatural}$ enjoys (*) and $\mathbb{E}[\mathsf{SampleNatural}(\mathcal{H}, \mathcal{B})] = \mathsf{R}_{(\mathcal{H},\mathcal{B})}$. By Lemma 2, the algorithm that constructs the $(D, \Sigma)$-synopsis $(\mathcal{H}, \mathcal{B})$ of $\bar{t}$, and runs $\mathsf{MonteCarlo}[\mathsf{SampleNatural}]$ with input $(\mathcal{H}, \mathcal{B})$, is an approximation scheme for $\mathsf{RelativeFreq}$. This algorithm is called $\mathsf{Natural}$.

**Symbolic Sampling Space.** An alternative sampler is one whose expected value, on input $(\mathcal{H}, \mathcal{B})$, is a number from which $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$ can be derived. We devise a slightly more complex sampling space, called symbolic, by exploiting ideas from [14]. Let $H_1, \dots, H_n$ be an arbitrary order-

[3]In case $\mathcal{H} = \emptyset$, we let $\mathsf{R}_{(\mathcal{H},\mathcal{B})} = 0$.

[4]In a data-efficient randomized approximation scheme for $\mathsf{EV}[\mathsf{Sample}]$ the output to be approximated is $\mathbb{E}[\mathsf{Sample}((\mathcal{H}, \mathcal{B}))]$, and the running time must be polynomial in $||\mathcal{H}, \mathcal{B}||$ (and $1/\epsilon$ and $\log(1/\delta)$), as these are the parameters of a synopsis that depend on the database.

ing (e.g., lexicographical) among the databases of $\mathcal{H}$. For each $i \in [n]$, $\mathcal{I}^i_{\mathcal{H},\mathcal{B}}$ is set of all $I \in \mathsf{db}(\mathcal{B})$ such that $H_i \subseteq I$. Our 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 $\mathsf{SampleKL}$, performs the following: *1)* sample $(i, I) \in \mathcal{S}^{\bullet}_{\mathcal{H},\mathcal{B}}$ u.a.r., *2)* if there is no $j < i$ s.t. $(j, I) \in \mathcal{S}^{\bullet}_{\mathcal{H},\mathcal{B}}$, then output 1, otherwise output 0. We can prove that $\mathsf{SampleKL}$ enjoys (*) and that its expected value is $r/|\mathcal{S}^{\bullet}_{\mathcal{H},\mathcal{B}}|$, where $r$ is the numerator of $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$. Hence, we consider the algorithm $\mathsf{KL}$, that constructs the $(D, \Sigma)$-synopsis $(\mathcal{H}, \mathcal{B})$ of $\bar{t}$, runs the algorithm $\mathsf{MonteCarlo[SampleKL]}$ with input $(\mathcal{H}, \mathcal{B})$, and multiplies its output by $|\mathcal{S}^{\bullet}_{\mathcal{H},\mathcal{B}}|/|\mathsf{db}(\mathcal{B})|$. By Lemma 2, $\mathsf{KL}$ is an approximation scheme for $\mathsf{RelativeFreq}$.

One can devise a slightly different sampler, called $\mathsf{SampleKLM}$, that has a lower variance in general w.r.t. $\mathsf{SampleKL}$. Using $\mathsf{SampleKLM}$ in place of $\mathsf{SampleKL}$, as discussed above, we obtain the approximation scheme for $\mathsf{RelativeFreq}$ called $\mathsf{KLM}$.

### 3.2. Union of Sets Approach

An approximation scheme, called *self-adjusting coverage algorithm* was presented in [14] for the *union of sets problem*, which takes a description of $n \geq 1$ sets $S_1, \ldots, S_n$, and the output is $|\bigcup_{i \in [n]} S_i|$. For space reasons, we do not present this algorithm, but only show how $\mathsf{RelativeFreq}$ reduces to this problem. We first construct the $(\Sigma, Q)$-synopsis of $D$ for $\bar{t}$, $(\mathcal{H}, \mathcal{B})$. Then, since $\bigcup_i \mathcal{I}^i_{\mathcal{H},\mathcal{B}}$ is the numerator of the ratio $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$, we approximate the value $|\bigcup_i \mathcal{I}^i_{\mathcal{H},\mathcal{B}}|$ via the self-adjusting coverage algorithm of [14], by seeing $(\mathcal{H}, \mathcal{B})$ as the description of the sets $\mathcal{I}^i_{\mathcal{H},\mathcal{B}}$, for each $i$. Then, dividing the result by $|\mathsf{db}(\mathcal{B})|$ will give an approximation of $\mathsf{R}_{(\mathcal{H},\mathcal{B})}$. This, together with Lemma 2, provides an approximation scheme for $\mathsf{RelativeFreq}$, which we dub $\mathsf{Cover}$.

## 4. The Benchmark

**Implementation.** Note that to compute the approximate relative frequency of each candidate tuple $\bar{t}$, the corresponding synopsis must be computed first, regardless of the chosen algorithm. We show we can construct the set $\mathsf{syn}_{\Sigma,Q}(D)$ of all pairs $(\bar{t}, S)$, with $S$ the $(D, \Sigma)$-synopsis of $\bar{t} \in \mathsf{dom}(D)^{|\bar{x}|}$, with $\mathsf{R}_{D,\Sigma,Q}(\bar{t}) > 0$, via a single SQL query, obtained by a careful rewriting of the original query $Q$. We call this initial precomputation of $\mathsf{syn}_{\Sigma,Q}(D)$ the *preprocessing step*. For implementing the approximation schemes, we extended the framework of [16] of approximation algorithms for the number of satisfying assignments of DNF formulas.

**Experimental Setting.** To effectively evaluate our algorithms, our test scenarios must expose how the running time of the approximation schemes is affected by key input parameters like the inconsistency of the database, and the number of joins in the query.

Data Generator. We generate databases using the dbgen tool of the TPC-H benchmark. Databases of varied size can be generated by providing a *scale factor* as input. Since the generated databases are consistent w.r.t. the underlying constraints we will later inject inconsistency via a noise generator that we developed.

Query-aware Noise Generator. General-purpose noise generators exist in the literature; see, e.g., [17]. However, all such tools are query-oblivious, i.e., do not take into account any query: 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 (and hence the synopses), as in general, databases are much larger than the actual synopses.

Given $D, \Sigma, Q(\bar{x})$, a percentage $0 < p < 1$ and an integer interval $[l, u]$, our noise generator constructs the set $\mathsf{syn}_{\Sigma,Q}(D)$, and for each $(\bar{t}, (\mathcal{H}, \mathcal{B}))$, randomly chooses $\ell = \lceil p \cdot |\mathcal{B}| \rceil$ blocks $B_1, \ldots, B_\ell$, and for each $B_i$, adds to $D$ a random number $t \in [l, u]$ of tuples having the same key value as the one of $B_i$. These tuples are constructed by reusing existing tuples from $D$, and by changing their key value. This ensures that the new tuples preserve the join patterns of $D$.

*Query Generator.* To generate our stress test queries we exploit a recent query generator [18], which we call *static query generator* (SQG) since it can control static query parameters, such as the number of joins, and the number of free variables (i.e., output variables). As we also want to precisely control the *size* of the synopsis, we also consider some dynamic query parameters.

Letting $\mathsf{syn}_{\Sigma,Q}(D) = \{(\bar{t}_i, (\mathcal{H}_i, \mathcal{B}_i))\}_{i \in [n]}$ for some $n \geq 1$, we consider 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 can affect any $(D, \Sigma)$-synopsis in $\mathsf{syn}_{\Sigma,Q}(D)$. Since synopses in $\mathsf{syn}_{\Sigma,Q}(D)$ can have different sizes, it is natural to consider the *average* size of a $(\Sigma, Q)$-synopsis in $\mathsf{syn}_{\Sigma,Q}(D)$ as a key parameter, which is given by $\frac{|\bigcup_{i=1}^{n} \mathcal{H}_i|}{|\mathsf{syn}_{\Sigma,Q}(D)|}$. We normalize the above to a number in the interval $(0, 1)$ by considering its inverse. We call this number the *balance of* $Q$ *w.r.t.* $D$. The closer the balance to 1 (resp., 0) is, the smaller (resp., larger) the average size of a $(\Sigma, Q)$-synopsis is.

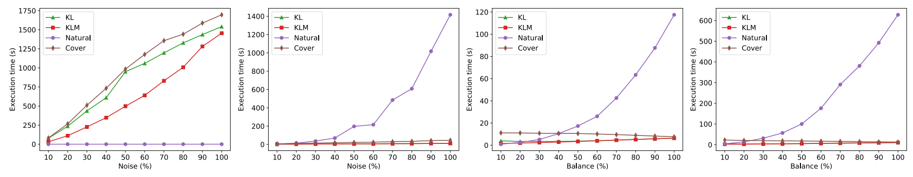
We developed a query generator, called dynamic, that modifies a query generated by the SQG so that it has a desired balance w.r.t. the given database $D$.

**Test Scenarios.** For our test scenarios we consider the TPC-H schema, denoted $\mathbf{S}_{\mathrm{H}}$, and the set $\Sigma_{\mathrm{H}}$ of primary keys over $\mathbf{S}_{\mathrm{H}}$ coming with the TPC-H benchmark. We generated a large set of database-query pairs $P_{\mathrm{H}}$ as follows: (1) First, we generated a (consistent) database $D_{\mathrm{H}}$ over $\mathbf{S}_{\mathrm{H}}$, using scale factor 1GB. The database contains roughly 9M tuples. (2) For each number of joins $j \in [5]$, we generated 5 base CQs $Q_j^i$, with $i \in [5]$, having $j$ joins, using SQG, producing a set $\mathcal{Q}$ of 25 CQs. Using our query-aware noise generator, for each $Q \in \mathcal{Q}$, we generated 10 inconsistent databases $D_Q[p]$, using noise levels $p \in \{0.1, 0.2, \ldots, 1\}$, and the same block interval $[2, 5]$. (4) Finally, for each $Q \in \mathcal{Q}$, and for each $D_Q[p]$, we generated 11 variations of $Q$, denoted $Q_p[q]$, having balance $q \in \{0, 0.1, 0, 2, \ldots, 1\}$ w.r.t. $D_Q[p]$, where balance $q = 0$ means the query $Q_p[q]$ is Boolean. $P_{\mathrm{H}}$ is then the set of pairs $\{(D_Q[p], Q_p[q]) \mid Q \in \{Q_j^i\}_{i,j \in [5]}, p \in \{0.1, \ldots, 1\}$ and $q \in \{0, 0.1, \ldots, 1\}\}$, containing 2750 database-query pairs.

We considered different scenarios (subsets of $P_{\mathrm{H}}$) by fixing all parameters but one. *Noise scenarios*: The sets $\mathsf{Noise}[q, j]$ of all pairs of $P_{\mathrm{H}}$, where the balance level ($q$) and the number of joins ($j$) are fixed. *Join scenarios*: The sets $\mathsf{Join}[p, q]$ of all pairs of $P_{\mathrm{H}}$, where the noise level ($p$) and the balance level ($q$) are fixed. *Balance scenarios*: The sets $\mathsf{Balance}[p, j]$ of all pairs of $P_{\mathrm{H}}$, where the noise level ($p$) and the number of joins ($j$) are fixed.

**Hardware and execution.** For the experiments we used two Desktop PCs each with an Intel(R) Core(TM) i5-8500 CPU@3.00GHz processor, 16GB RAM, 500GB mechanical drive, running Xubuntu 19.04 64-bit. All databases are stored in a PostgreSQL 11.5 instance. The approximation schemes and the preprocessing step are implemented in C++ and SQL, respectively.

We fixed $\delta = 0.25$ and $\epsilon = 0.1$, i.e., 75% confidence and 10% error. We report the time required to execute each algorithm over the synopses of *all* output tuples, excluding the time of the preprocessing step since it is the same for all the approximation schemes. Each data point



(a) Noise$[0,3]$ (b) Noise$[0.5,3]$ (c) Balance$[0.2,3]$ (d) Balance$[0.4,3]$

**Figure 1:** Noise and balance scenarios - Noise$[balance, joins]$, and Balance$[noise, joins]$

in a plot of Fig. 1 is the average runtime over all the CQs for that X-axis value; due to space constraints, here we report only some representative noise and balance scenarios (see Fig. 1).

## 5. Take-home Messages

Our analysis reveals a striking difference between Boolean and non-Boolean CQs.

**The Boolean Case.** Here, Natural is the best performer, no matter the noise and 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. This is mainly due to the fact that for Boolean queries, the (only) synopsis is very large in general, and thus the relative frequency is very high. Hence, since Natural directly estimates the value of the relative frequency via the natural sampling space, requires less samples to obtain a good estimate.

**The Non-Boolean Case.** In this case, 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. The reson for this outcome is that for higher values of balance, the average size of a synopsis is low, which then implies that the relative frequency of the corresponding tuple is usually close to 0, hence, sampling from the symbolic space helps avoiding to perform a large number of samples like Natural does. Moreover, the low-variance sampler of KLM lets the algorithm find the optimal number of samples earlier than KL, on average.

**Practical Applicability.** In most of our experiments, the preprocessing step took less than 30 seconds. Furthermore, for modest scenarios, which is what we expect to face in practice, the running time of the best performing approximation scheme is in the order of seconds. We also considered larger databases, up to 120M tuples (15GBs), for which the overall running time of the preprocessing step is encouraging (considering the weak machines used): it is in the order of minutes, while the best approximation algorithm for each case always performs in the order of seconds. Thus, we believe applying approximate CQA in practice is not an unrealistic goal.

## Acknowledgments

Part of Calautti's and Console's work was done while they were research associates at the University of Edinburgh. Console has been partially supported by MIUR under the PRIN 2017 project “HOPE” (prot. 2017MMJJRE), and by the EU under the H2020-EU.2.1.1 project TAILOR, grant id. 952215. Pieris was supported by the EPSRC grant EP/S003800/1 “EQUID”.

## References

[1] M. Arenas, L. E. Bertossi, J. Chomicki, Consistent query answers in inconsistent databases, in: PODS, 1999, pp. 68–79.

[2] M. Calautti, M. Console, A. Pieris, Counting database repairs under primary keys revisited, in: PODS, 2019, pp. 104–118.

[3] P. Koutris, J. Wijsen, Consistent query answering for primary keys in logspace, in: ICDT, 2019, pp. 23:1–23:19.

[4] B. ten Cate, G. Fontaine, P. G. Kolaitis, On the data complexity of consistent query answering, Theory Comput. Syst. 57 (2015) 843–891.

[5] S. Greco, C. Molinaro, I. Trubitsyna, Computing approximate query answers over inconsistent knowledge bases, in: IJCAI, 2018, pp. 1838–1846.

[6] A. A. Dixit, P. G. Kolaitis, A sat-based system for consistent query answering, in: SAT, 2019, pp. 117–135.

[7] A. Fuxman, E. Fazli, R. J. Miller, Conquer: Efficient management of inconsistent databases, in: SIGMOD, 2005, pp. 155–166.

[8] S. Greco, C. Molinaro, Probabilistic query answering over inconsistent databases, Ann. Math. Artif. Intell. 64 (2012) 185–207.

[9] M. Calautti, L. Libkin, A. Pieris, An operational approach to consistent query answering, in: PODS, 2018, pp. 239–251.

[10] D. Maslowski, J. Wijsen, A dichotomy in the complexity of counting database repairs, J. Comput. Syst. Sci. 79 (2013) 958–983.

[11] N. Fiorentino, S. Greco, C. Molinaro, I. Trubitsyna, ACID: A system for computing approximate certain query answers over incomplete databases, in: SIGMOD, 2018, pp. 1685–1688.

[12] S. Greco, C. Molinaro, I. Trubitsyna, Approximation algorithms for querying incomplete databases, Inf. Syst. 86 (2019) 28–45.

[13] M. Console, P. Guagliardo, L. Libkin, E. Toussaint, Coping with incomplete data: Recent advances, in: PODS, 2020, pp. 33–47.

[14] R. M. Karp, M. Luby, N. Madras, Monte-carlo approximation algorithms for enumeration problems, J. Algorithms 10 (1989) 429–448.

[15] P. Dagum, R. M. Karp, M. Luby, S. M. Ross, An optimal algorithm for monte carlo estimation, SIAM J. Comput. 29 (2000) 1484–1496.

[16] K. S. Meel, A. A. Shrotri, M. Y. Vardi, Not all FPRASs are equal: Demystifying FPRASs for DNF-counting, Constraints 24 (2019) 211–233.

[17] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, D. Santoro, Messing up with BART: Error generation for evaluating data-cleaning algorithms, PVLDB 9 (2015) 36–47.

[18] M. Benedikt, G. Konstantinidis, G. Mecca, B. Motik, P. Papotti, D. Santoro, E. Tsamoura, Benchmarking the chase, in: PODS, 2017, pp. 37–52.