Semantics-Aware Estimation of Number of Answers to Conjunctive Queries (With Experimental Details)

Abstract

The problem of estimating the number of answers to a conjunctive query is fundamental in data management. In current systems it is addressed via a combination of syntactic analysis of the formula and statistics on the data. We look at how to use the semantics of data to improve the precision of estimates. Our framework allows one to layer a search for semantically-equivalent queries ("rewritings") on top of any existing technique for semantics-unaware estimation. We couple this with methods for exploiting semantic information within the estimation of individual rewritings.

1 Introduction

This work considers the following problem: given a *query*, given as an open formula in logic, we want to estimate the number of results in its evaluation. This estimation is a crucial component in coming up with an efficient evaluation plan for a query within database management systems (DBMS's). Indeed the difficulty in getting accurate estimates has been a chief impediment to progress in efficient evaluation. Estimation involves at least two distinct tasks. First, one must derive and maintain *statistics* representing information about the size of parts of the dataset; these could be simply cardinalities of relations or columns, or finer-grained information such as histograms. Secondly one must *estimate the size of query output using the statistics*. The standard methods for estimating query size based on statistics work syntactically: via recursion, bottom-up on the syntax tree representing the query. For each connective, there is a corresponding rule estimating its output in terms of its input, with the base case being given by the statistics. Implicit in these rules are a number of strong assumptions on the distribution of data within a single relation, or the independence of data in two relations. It has long been realized that these assumptions can lead to large query size estimation errors (e.g. [Christodoulakis, 1984] and [Ioannidis, 2003]). It is likewise well-known that size estimation errors propagate exponentially through conjunctions [Ioannidis and Christodoulakis, 1991] and that size estimation errors have an enormous impact on optimization [Reddy and Haritsa, 2005; Leis *et al.*, 2015]. While there has been significant progress in enhancing the state of the art, research has generally focused on special cases: e.g., estimating the size of a conjunction using histograms on the conjuncts.

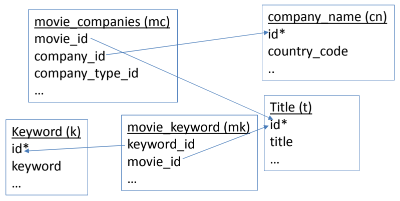

Figure 1: Fragment of IMDB schema

An obvious question is how to use semantic information to limit or even eliminate the use of assumptions that play such a large role in syntax-driven approaches. Special cases of this problem have been considered before in the data management community; for instance, algorithms that attempt to rewrite the query so that outpt size can be calculated easily using a set of views [Bruno and Chaudhuri, 2002]. A more ambitious approach is to see the problem as a special case of query answering for aggregate queries over incomplete data and ontological rules [Kostylev and Reutter, 2015; Calvanese *et al.*, 2008]; or as a validity problem via powerful logics with counting quantification [Pratt-Hartmann, 2009]. However, even powerful formalisms such as [Pratt-Hartmann, 2009] cannot deal with arbitrary arity schemas or express common statistical summaries such as histograms, while at the same time the complexity of the corresponding query answering problems is quite high. In this paper we consider a middle ground; unlike work in the database community [Bruno and Chaudhuri, 2002; 2004] we describe a method to incorporate reasoning from a broad class of horn rules into size estimation. Our method does not require a reasoning system that works natively with aggregation; further we can work *on top of* existing estimation approaches, including heuristic ones. Our approach will not provide provably tight upper bounds in all cases, but allows us to dramatically improve the estimates given by more specialized approaches.

Example 1 *We use an example from the IMDB benchmark for query optimization of [Leis* et al.*, 2015], shown in Fig. 1, with key-to-foreign-key rules as arrows. These semantic relationships can be read from an SQL schema, and captured* *in common knowledge representation formalisms (e.g. existential rules, see Sec. 2). We consider query $Q(text)$ given by $\exists cid, mcid, mid, kid\ cn(cid, \mathsf{d}) \wedge mc(mcid, cid, mid) \wedge k(kid, \mathsf{c}) \wedge mk(mid, kid) \wedge t(mid, text)$, with c, d constants.*

Commercial DBMS's, e.g., SQL Server, significantly misestimate the size of Q. This relates to the assumption of independence between the tuples of relation k having c in column two and those that overlap with relation mk on the first column.

Suppose the datasource maintains a relation V as a materialized view, *meaning that the DBMS enforces that V will always satisfy a corresponding* view definition *in terms of other relations. The view definition is expressed in logic as $V(mcid) \leftrightarrow cn(cid, \mathsf{d}) \wedge mc(mcid, mid, cid) \wedge mk(mid, kid) \wedge k(kid, \mathsf{c}) \wedge mc(mcid, mid, cid)$.*

Using the semantic information in an SQL schema, our system can determine that Q can be rewritten as $Q' = \exists cid, mcid, mid\ mc(mcid, cid, mid) \wedge t(mid, text) \wedge V(mcid)$. We can reason that the size of V is a guaranteed bound on the size of Q, not relying on any independence assumptions or heuristics. $\square$

In the examples above we could use rules and catalog information to get *hard upper bounds*. For example, we can determine that the size of V is a guaranteed bound on the size of the output Q, not relying on any independence assumptions or heuristics. But we will see that even when we cannot get hard upper bounds using any rewriting, our system will be able to *combine multiple rewritings, reasoning, and heuristics to get better estimates*.

Example 2 *The following example over the TPC-H schema shows the benefit of semantics-based rewritings, even when we need assumptions to estimate the size of any rewriting. Consider query*

$$Q_{tpch}(lkey, lno, okey, pkey) =$$
$$lineitem(lkey, lno, pkey, \mathsf{n}, \mathsf{o}, \mathsf{tb}, \mathsf{ml}) \wedge$$
$$orders(okey, price) \wedge part(pkey, price)$$

with constants n, o, tb and ml. Assume there are views $V_1(lkey, lno, pkey)$, V_2, $(okey, pkey)$ defined by the existential rules

$$V_1(lkey, lno, pkey) \leftrightarrow$$
$$item(lkey, lno, pkey, \mathsf{n}, \mathsf{o}, \mathsf{tb}, \mathsf{ml}) \wedge part(pkey, price)$$
$$V_2(okey, pkey) \leftrightarrow orders(okey, price) \wedge part(pkey, price)$$

A commercial DBMS will automatically generate histograms on all the base relations, including $orders$ and $part$, as well as on views V_1, V_2, but will ignore the rules extracted from the schema for estimating the size of Q_{tpch}. Such systems return estimates one order of magnitude too high. This is due to the fact that the default histograms they build on the second positions of $orders$ and $part$ are not accurate enough to estimate the size of the conjunction. For example, due to these histograms, SQL Server estimates the size of $\exists okey, pkey\ orders(okey, price) \wedge part(pkey, price)$ as $41,337$ while its actual size is $9,029$.

Making use of all extracted rules from the schema and the underlying statistics, we cannot derive a tight upper bound. However, using the extracted rules we can rewrite the query as ${Q'}_{tpch} = V_1(lkey, lno, pkey) \wedge V_2(okey, pkey)$, and we can see from the base statistics that the size estimate of this rewriting has higher quality than an estimate of the initial query: in estimating the size of ${Q'}_{tpch}$ an optimizer will use histograms on the relevant positions of V_1 and V_2 to estimate the conjunction size, and in contrast to the histograms on the second positions of $orders$ and $part$, the buckets for V_1 and V_2 have lower variance. Our system can identify ${Q'}_{tpch}$ as the best rewriting, giving a five-fold improvement in accuracy. $\square$

We present SEUSS (Size Estimation Using Semantics and Statistics), our framework for estimating query output size. SEUSS is based on a search algorithm that enumerates containing rewritings under semantic rules. In the most basic instantiation, the search algorithm enumerates containing rewritings and then calls a "black-box" semantics-unaware size-estimation algorithm on each rewriting. For example, the black-box could be a classical size-estimation procedure working bottom-up on the syntax of the rewriting. But we provide more refined algorithms that also make use of rules in estimating a single rewriting. The idea of searching through a space of *exact* rewritings is reminiscent of the chase and backchase technique that has been used for semantics-based query optimization in databases [Deutsch *et al.*, 2006; Popa, 2000] and in KR [Toman and Weddell, 2011]. We show that the idea can be useful (perhaps even more so) in size estimation.

Our first "white-box algorithm" for estimating individual rewritings returns size estimates that are provable upper bounds for the output size of a given rewriting assuming the size estimates are accurate – what we call "hard upper bounds". We build on work of [Khamis *et al.*, 2016], to obtain a bound which incorporates both key constraints, statistical views, and information extracted from histograms. We also provide an algorithm for estimating rewritings that uses reasoning but allows more extensive use of heuristics, distinguishing better and worse estimates.

We provide an extensive empirical evaluation, giving evidence that SEUSS provides benefits over state-of-the-art systems for query estimation. The evaluation considers a diverse set of scenarios, giving insight on the behavior of the estimates produced by SEUSS in absolute terms and in comparison to commercial systems accessing the same statistical information. Since a major application of size estimation is in deriving efficient query plans, our experiments also analyze the runtime impact of the resulting estimates, using the most established benchmark for query optimization. *In summary, to the best of our knowledge this work gives the first means of enhancing statistics with a broad use of reasoning for cardinality estimation. Our work also represents the first approach to semantics-based estimation that is modular, readily adaptable to richer classes of rules. Our evaluation provides the most comprehensive look at exploiting semantic information in query size estimation in practice.*

Organization. After summarizing related work we formalize the problem definition. We overview the basic theoretical tools used in our analysis, all of which involve the analysis of queries under rules (Subsection 2.1). We provide our high-level framework for estimation based on semantics (Section 3), and complete our system description with algorithms for estimating a single rewriting (Section 4). The remainder of the text is concerned with explaining our implementation, benchmark, and experimental evaluation.

2 Preliminaries

The inputs to our size estimation problem are a schema, statistical information, semantic rules, and a query. We start with a standard relational schema, given by a set of *relations*. A relation R has an *arity*: intuitively a relation with arity n stores n-tuples. We will move back and forth between the *named perspective* on relations — relation movie_keyword has attributes movie_id and keyword_id — and the *positional perspective* — relation movie_keyword consists of pairs, where the first element of the pair represents the movie_id and the second element represents the keyword_id. The positional perspective can also be thought of as the assumption that our relations have attributes $\#1 \ldots \#n$ for some number n, and refer to the attributes as *positions*. Within the schema we distinguish a subset of the relations as *estimatable relations*. Informally, these represent data about which we have statistics available. For example, views defined over a set of base relations. Since views can be subsumed via rules, The framework subsumes the case where we have statistics on views. Each estimatable relation is associated with a size upper bound or a histogram on some set of positions in the relation, including at least every individual position. Informally, $\mathsf{upper}(\mathsf{StatTab})$ is an upper bound on the number of tuples in $\mathsf{StatTab}$. We do not assume statistics are accurate, but instead allow quantitative information to be stored on their quality (e.g. variance and sampling factor for histograms). For histograms we store the variance and sampling factor. For example, for base relations that do not have any corresponding histogram built, the upper bound might be a default, and thus given a low score. We focus on user queries that are expressed as *conjunctive queries* (CQs), written in logical notation as $\exists x_1, \ldots, x_n\; A_1 \wedge \ldots \wedge A_n$, where the A_i are *relational atoms*, of the form $R(\vec{t})$ where $\vec{t}$ is an $\mathsf{arity}(R)$-tuple of variables or constants. The variables in the query other than x_i are the *free variables*.

Our goal is to make use of *semantic information* relating the relations in the schema. We will consider *existential rules* or *Tuple Generating Dependencies* (TGDs), logical sentences of the form

$$\forall \vec{x}\; \lambda(\vec{x}) \rightarrow \exists \vec{y}\; \rho(\vec{x}, \vec{y}) \qquad (1)$$

where $\lambda(\vec{x})$ is a conjunction of relational atoms whose free variables are contained in $\vec{x}$, and $\rho(\vec{x}, \vec{y})$ is a conjunction of relational atoms whose free variables are contained in $\vec{x} \cup \vec{y}$. We also consider *Equality Generating Dependencies* (EGDs), sentences of the form (2), $\forall \vec{x}\; \lambda(\vec{x}) \rightarrow x_i = x_j$ where $\lambda(\vec{x})$ is a conjunction of relational atoms over variables $\vec{x}$.

$$\forall \vec{x}\; \lambda(\vec{x}) \rightarrow x_i = x_j \qquad (2)$$

The left-hand side of a TGD or an EGD (i.e., the conjunction $\lambda(\vec{x})$) is the *body* of the dependency, and the right-hand side is the *head*. By a slight abuse of notation, we often treat heads and bodies as sets of atoms. For brevity, we commonly omit the leading universal quantifiers in dependencies. These include SQL's *key and foreign key constraints* and TGDs stating that a view relation corresponds to its definition. Returning to Example 1, the fact that a tuple satisfying the view definition of V must be contained in V could be expressed as a TGD, which we denote as DefToV:

$$cn(cid, \mathsf{d}) \wedge mc(mcid, mid, cid) \wedge \\ mk(mid, kid) \wedge k(kid, \mathsf{c}) \rightarrow V(mcid)$$

The statement that tuples in the view relation V must match the definition would be expressed as the following rule, denoted VToDef

$$V(mcid) \rightarrow \\ \exists cid\; kid\; mid\; cn(cid, \mathsf{d}) \wedge mc(mcid, mid, cid) \wedge \\ mk(mid, kid) \wedge k(kid, \mathsf{c})$$

Given a schema, estimatable relations, and semantic rules, along with a CQ Q, our goal is to estimate the size of Q.

Semantics of queries and rules. An *instance* I of a schema assigns to each n-ary relation R in the schema a set $I(R)$ of n-tuples of values. We distinguish between values that can make up data instances, and which satisfy the unique name assumption: These will be denoted as *constants*, and the set of constants is abbreviated Const. Other values can be generated within our reasoning algorithms – these are known as *nulls*. An atom is *ground* (or a *fact*), if it does not contain variables. Thus a fact is of the form $R(v_1 \ldots v_n)$ where R is of arity n and $v_1 \ldots v_n$ are values. An instance can equivalently be seen as a set of relational facts. A *homomorphism* of a CQ Q to a database I is a mapping σ of the variables in Q to domain elements of I such that for each atom $R(w_1 \ldots w_k)$ in Q, $R(\sigma(w_1) \ldots \sigma(w_k)) \in I$, where we extend σ to constants by the identity. The *solutions* to Q in I, denoted $Q(I)$, are the restrictions of homomorphisms of Q to the free variables.

Let I be an instance and τ be a TGD of the form (1) or an EGD of the form (2). The notion of I *satisfying* τ can be stated using homomorphisms. We say that TGD τ is satisfied in I, written $I \models \tau$, if every homomorphism of the body λ extends to a homomorphism of the head ρ. Similarly an EGD of form (2) is satisfied if for every homomorphism σ of λ we have $\sigma(x_i) = \sigma(x_j)$.

Given a set of rules Σ, a rule σ is *inferred from* Σ is every instance that satisfies Σ satisfies σ.

2.1 The chase and semantics-based rewritings

Our high-level algorithm explores a space of formulas logically equivalent to the original query Q, looking for ones that assist in cardinality estimation. Because we need to account for (and utilize) the semantic rules Σ, we may need to consider formulas including relations not mentioned in Q. More precisely we will be interested in Σ*-relative containing rewritings* (Σ-RCRs, for short): conjunctive query Q' is an Σ-RCR of conjunctive query Q if for every instance I that satisfies Σ, $Q(I) \subseteq Q'(I)$. We will be particularly interested in Σ*-relative equivalent rewritings* of Q (for short Σ-RER), meaning that $Q'(I) = Q(I)$ for any instance I satisfying the rules in Σ.

The fundamental computational problem of finding Σ-RCRs and Σ-RERs for a conjunctive query Q, when the rules are TGDs and EGDs, can be solved using the *chase method* [Maier *et al.*, 1979; Fagin *et al.*, 2005]. Informally the method builds "all of the consequences of Q", and uses these facts to determine which queries are Σ-RCRs of Q and which are Σ-RERs of Q.

The chase is a process that transforms an initial instance by a sequence of *chase steps* until all dependencies are satisfied. A *trigger* for a dependency in an instance I is a homomorphism of the body of the dependency to I. Given an instance I, a chase step for a TGD τ refeq:tgd-form and a trigger h extends I with new facts of the form $h'(A_i)$, where A_i is an atom of the head σ, h' is a substitution such that $h'(x_i) = h(x_i)$ for each variable $x_i \in \vec{x}$, and $h'(y_j)$ for each $y_j \in \vec{y}$ is a fresh value (a "null") that does not occur in I. A chase step for an EGD and a trigger h merges $h(x_i)$ with $h(x_j)$. For Σ a set of TGDs and EGDs and I a finite instance, a *chase sequence* for Σ and I is a sequence $I_0, I_1, \ldots$ such that $I = I_0$ and, for each $i \geq 0$, if I_{i+1} exists it is obtained from I_i by applying a chase step to I_i. A chase sequence is *terminating* if it reaches an instance where the rules are all satisfied. It is known that for many common sets of dependencies Σ, such as those corresponding to view definitions, any instance I has a chase sequence for Σ that terminates. The *result* of a terminating chase sequence is then the final instance. We denote an arbitrary result of a terminating chase sequence starting at I, applying dependencies from Σ, as $\mathsf{Chase}_\Sigma(I)$.

The chase gives a means for inferring new facts about a database instance. It can also be applied to infer consequences of a query, by "turning the query into an instance". Given a conjunctive query Q, the *canonical database of* Q, abbreviated $\mathsf{CDB}(Q)$, is the database in which every variable v of the query is turned into a null e_v, and every atom of Q is turned into a fact by changing any variable v in the fact to e_v.

We apply the chase and the canonical database construction to solve several problems. First, the chase gives us a simple test of whether a CQ Q' is a Σ-RCR of Q:

Proposition 1 *[Maier* et al.*, 1979] Q' is a* Σ-RCR *of Q exactly when there is a* match of Q' in a chase sequence from $\mathsf{CDB}(Q)$*: that is, a homomorphism which maps each free variable x of Q' to constant e_x.*

We are interested in Σ-RCR's that use only estimatable relations. Proposition 1 implies that these are exactly the CQs over the estimatable relations that have a match in $\mathsf{Chase}_\Sigma(\mathsf{CDB}(Q))$. A set of atoms $\mathcal{F}$ in $\mathsf{Chase}_\Sigma(\mathsf{CDB}(Q))$ is said to *cover the free variables of* Q if for every free variable v of Q, $\mathcal{F}$ contains an atom with the corresponding constant e_v. For any set of atoms $\mathcal{F}$ in $\mathsf{Chase}_\Sigma(\mathsf{CDB}(Q))$ let $\mathsf{QueryOf}(\mathcal{F})$ be the conjunctive query formed by turning all nulls of $\mathcal{F}$ into existentially quantified variables and value e_v back to a free variable v. Proposition 1 implies that if $\mathcal{F}$ covers the free variables of Q, $\mathsf{QueryOf}(\mathcal{F})$ is a Σ-RCR. The same proposition can be used to show that every Σ-RCR is implied by a Σ-RCR of the form $\mathsf{QueryOf}(\mathcal{F})$. We will therefore focus on rewritings of the form $\mathsf{QueryOf}(\mathcal{F})$ for $\mathcal{F}$ a subset of $\mathsf{Chase}_\Sigma(\mathsf{CDB}(Q))$: *chase-based rewritings*.

Our second use of the chase is to determine which Σ-RCR's are equivalent rewritings. By Prop. 1, this holds of a Σ-RCR Q' exactly when there is a match of Q into $\mathsf{Chase}_\Sigma(\mathsf{CDB}(Q'))$. We will be interested in the case where Q' is already a chase-based rewriting, in which case the characterization of Σ-RERs has a simpler form:

Proposition 2 *If Q' is a chase-based rewriting of the form* $\mathsf{QueryOf}(\mathcal{F})$ *for* $\mathcal{F} \subseteq \mathsf{Chase}_\Sigma(\mathsf{CDB}(Q))$ *then Q' is a* Σ-RER *of Q exactly when Q has a match in* $\mathsf{Chase}_\Sigma(\mathcal{F})$.

Finally, we apply the chase to see whether a TGD or EGD σ is *inferred* from a set of TGDs and EGDs Σ. This means that in any instance I satisfying each rule in Σ, I satisfies σ. We use the following well-known characterization of inference [Beeri and Vardi, 1984]:

Proposition 3 *An EGD $\forall \vec{x}\ \lambda(\vec{x}) \rightarrow x_i = x_j$ is inferred from a set of TGDs and EGDs Σ exactly when there is a chase sequence of* $\mathsf{CDB}(\lambda)$ *with respect to Σ that leads to a chase step identifying e_{x_i} and e_{x_j}. A TGD*

$$\forall \vec{x}\ \lambda(\vec{x}) \rightarrow \exists \vec{y}\ \rho(\vec{x}, \vec{y}) \tag{3}$$

is inferred from Σ exactly when there is a chase sequence of $\mathsf{CDB}(\lambda)$ *that leads to an instance where $\exists \vec{y}\ \rho(\vec{x}, \vec{y})$ has a match.*

3 Size Estimation Framework

The chase-based methods outlined previously underlie our strategy for estimating the size of Q. Compute $\mathsf{Chase}_\Sigma(\mathsf{CDB}(Q))$, then compute subsets of facts $\mathcal{F}$ from $\mathsf{Chase}_\Sigma(\mathsf{CDB}(Q))$, where each fact from $\mathcal{F}$ comes from an estimatable relation. If a subset of facts $\mathcal{F}$ is a Σ-RCR, compute $\mathsf{QueryOf}(\mathcal{F})$ and estimate its size. Finally, use the "most accurate" size estimate to approximate the size of Q. Our algorithm requires in the input a method SizeOf for estimating the size of a query and a method $\mathsf{QualityOf}$ for estimating the "quality" of the estimate. We defer the discussion of both methods at Section 4, but note that we will consider whether a query is a Σ-RER in quality evaluation.

We describe a way of building up chase-based rewritings based on subsets of facts $\mathcal{F}$ inductively in rounds, analogous to the dynamic programming algorithms used in traditional query optimization. In doing this, we annotate each subset $\mathcal{F}$ with its consequence facts $\mathsf{Chase}_\Sigma(\mathcal{F})$. This is done in order to assess whether $\mathcal{F}$ is an Σ-RCR or an Σ-RER of Q.

Definition 1 (Annotated rewriting) *An annotated rewriting $\mathcal{R}$ consists of a collection of facts $\mathcal{F} \subseteq$* $\mathsf{Chase}_\Sigma(\mathsf{CDB}(Q))$ *using estimatable relations, along with* $\mathsf{Chase}_\Sigma(\mathcal{F})$*. $\mathcal{F}$ is referred to as the* accessed facts *while the facts in* $\mathsf{Chase}_\Sigma(\mathcal{F})$ *are the* annotations*. The composition of two annotated rewritings $\mathcal{R}_i$ and $\mathcal{R}_j$ denoted $\mathcal{R}_i \circ \mathcal{R}_j$, is a new annotated rewriting with accessed facts $\mathcal{F}_i \cup \mathcal{F}_j$ and annotations* $\mathsf{Chase}_\Sigma(\mathcal{F}_i \cup \mathcal{F}_j)$*, where $\mathcal{F}_i$ and $\mathcal{F}_j$ are the accessed facts of $\mathcal{R}_i$ and $\mathcal{R}_j$, respectively.*

We often identify a rewriting with the set of accessed facts $\mathcal{F}$, writing $\mathcal{R}(\mathcal{F})$. We are now ready to present our semantics-aware size estimation algorithm (see Algorithm 1): first compute all *atomic annotated rewritings*, where $\mathcal{F}$ consists of a single fact from $\mathsf{Chase}_\Sigma(Q)$ whose relation is estimatable. In round $k+1$ create new annotated rewritings composing rewritings annotated computed in previous rounds.

The number of possible rewritings can be quite large, and we can restrict the space to make exploration more efficient. In our implementation we only compose with one atomic annotated rewriting on the right. It is easy to show that if there is any rewriting using the estimatable relations, there is one of this form.

The following is easy to verify:

Proposition 4 *For any rewriting length limit d Algorithm 1 will estimate rewritings equivalent to* $\mathsf{QueryOf}(\mathcal{F})$ *for all subsets $\mathcal{F}$ of* $\mathsf{Chase}_\Sigma(\mathsf{CDB}(Q))$ *of size at most d.*$\square$

We employ different techniques to improve the performance of Algorithm 1. We restrict the search space so as not to compose two rewritings where one is semantically equivalent to a subquery of the other. We can check this using Proposition 1, avoiding the creation of these "redundant" compositions. As chasing is an expensive operation, we perform the chasing involved in forming the annotations for different compositions within a round in parallel.

Algorithm 1: Semantics-aware size estimation

```
Input:  query $Q$, rules $\Sigma$, rewriting limit $d$
Output:  size estimate of $Q$
1  Initialize R with atomic rewritings from
   $\mathsf{Chase}_\Sigma(\mathsf{CDB}(Q))$ based on estimatable relations;
2  Initialize B with any non-($\mathsf{SizeOf}$, $\mathsf{QualityOf}$) dominated
   rewritings of R that covers the free variables of $Q$;
3  $r := 1$;
4  while $r \leq d$ and R has changed do
5      for $\mathcal{R}_1, \mathcal{R}_2 \in$ R with $\mathcal{R}_2$ atomic do
6          Add $\mathcal{R} := \mathcal{R}_1 \circ \mathcal{R}_2$ to R;
7          Let $\mathcal{F}$ be the accessed facts of $\mathcal{R}$;
8          if $\mathcal{F}$ covers the free variables of $Q$ and
           $\mathsf{QueryOf}(\mathcal{F})$ is not ($\mathsf{SizeOf}$, $\mathsf{QualityOf}$)
           dominated by any rewriting in B then
9              $D :=$ rewritings in B with $\mathsf{SizeOf}$ or
               $\mathsf{QualityOf}$ dominated by $\mathsf{QueryOf}(\mathcal{F})$;
10             B $:=$ B $\cup \{\mathsf{QueryOf}(\mathcal{F})\} - D$;
11     $r := r + 1$;
12 return median value of $\mathsf{SizeOf}(Q')$ over $Q' \in$ B
```

Example 3 *We step through Algorithm 1 for Example 1. The canonical database of Q* $\mathsf{CDB}(Q)$ *consists of the facts:* $cn(e_{cid}, \mathsf{d}), mc(e_{mcid}, e_{cid}, e_{mid}), mk(e_{mid}, e_{kid}), k(e_{kid}, \mathsf{c})$, $t(e_{mid}, e_{text})$. *The constants correspond to variables of the query, with* e_{text} *corresponding to its free variable. Now, due to the rules defining view* V, $\mathsf{Chase}_\Sigma(\mathsf{CDB}(Q))$ *will add the fact* $V(e_{mcid})$ *to* $\mathsf{CDB}(Q)$.

Assuming we have statistics on all relations, our initial round includes rewritings computed from each single fact in $\mathsf{Chase}_\Sigma(\mathsf{CDB}(Q))$. *Only one of these first-round rewritings, namely* $\mathcal{R}(t(e_{mid}, e_{text}))$, *contains the value* e_{text} *corresponding to the free variable of the query. Hence only this rewriting is a* Σ-RCR, *and we need only to estimate the size and the quality of the corresponding query, based on our size estimate for t.*

In the second round we consider all pairs of rewritings computed in round 1. All pairs containing $t(e_{mid}, e_{text})$ *will correspond to* Σ-RCR *rewritings, and thus we will estimate the size and quality of the corresponding queries. We can check that none of these is an* Σ-RER, *and thus no round* 1 *or* 2 *queries will be a* Σ-RER.

In round 3 *we consider* $\mathcal{R} = \mathcal{R}' \circ \mathcal{R}(V(e_{mcid}))$ *where* $\mathcal{R}'$ $= \mathcal{R}(t(e_{mid}, e_{text})) \circ \mathcal{R}(mc(e_{mcid}, e_{cid}, e_{mid}))$. *The query built from the accessed facts of* $\mathcal{R}'$ *has a match for Q. Thus this rewriting is a* Σ-RER, *and will be given preference over the other estimates for Q.* $\square$

4 Estimating a single rewriting

We could in principle use any semantics-unaware method for estimating the size of the rewritings computed during the iterations of Algorithm 1. Here, we will make use of semantic information to also improve estimation *within a single rewriting*. The idea will be to make use of inferred rules that allow better cardinality information.

Rule inference. A component in all of our rewriting estimation techniques is *rule inference*: determining rules that hold on relations (e.g. estimatable relations) from a given set of input rules. As mentioned in Section 2 rule inference for TGDs and EGDs can be performed via the chase.

Hard bounds on the size of a rewriting. We begin with a semantics-aware technique that produces a *hard upper bound* on the size of a rewriting given accurate statistics and without making use of any approximations, independence assumptions, or heuristics. This technique is called CLLP ([Khamis *et al.*, 2016]) and is a variation of the "AGM bound", a technique for estimating the size of a CQ [Atserias *et al.*, 2013; Gottlob *et al.*, 2012; Ngo *et al.*, 2014; Joglekar and Ré, 2016; Khamis *et al.*, 2016]. The CLLP extension takes into account finer statistics, *degree bounds*. In simplified form: if there is a query atom $R(\vec{w})$ containing variables Y, $X \subseteq Y$, and for each binding X, there are at most k values for Y on matching R-tuples, then this implies there is a degree bound k for (X, Y). These bounds can either be data-independent (e.g. FDs) or derived from statistics CLLP uses them to create a linear programming (LP) problem, such that any optimal solution represents a bound on the size of Q.

A simplified version of CLLP has non-negative integer variables $h_{\mathbf{X}}$ for every set of variables in the query. We want to maximize $2^{h_{\mathbf{O}}}$ subject to the LP constraints below, where $\mathbf{O}$ is the set of free variables of the query.

- (Degree bound) $h_{\mathbf{Y}} - h_{\mathbf{X}} \leq log(k)$, where k is a degree bound for $X \subseteq Y$
- (Monotonicity) $h_{\mathbf{X}} - h_{(\mathbf{X} \cup \{\mathbf{x}\})} \leq 0$ for subset $\mathbf{X}$ and single variable x
- (Submodularity) $h_{(\mathbf{S} \cup \{x,y\})} + h_{\mathbf{S}} - h_{(\mathbf{S} \cup \{x\})} - h_{(\mathbf{S} \cup \{y\})} \leq 0$ for every set of variables $\mathbf{S}$ and each $x \neq y$ variables not in $\mathbf{S}$

[Khamis *et al.*, 2016] shows that the solution to the LP is an upper bound on query size. Note that unlike in earlier work on the AGM bound [Atserias *et al.*, 2013; Gottlob *et al.*, 2012], there are exponentially many inequalities. Again following [Khamis *et al.*, 2016] we can decrease this by restricting to subsets X, S closed under FDs of the schema, and adding a closure operation under FDs in the indexing expressions above (e.g. replacing $X \cup \{x\}$ with its closure); we can also restrict monotonicity to sets X containing the free variables of the query. Using these optimizations we can handle all queries within standard benchmarks even using an opensource LP solver. CLLP is not able to handle TGDs, and thus the use of our our high-level algorithm and FD inference on top of it will be crucial (see Section 5).

Example 4 *Recall the rewriting* Q' *from Example 1:*

$$\exists cid, mcid, mid\; mc(mcid, cid, mid) \wedge t(mid, text) \wedge V(mcid)$$

We will explain how CLLP *derives* $|V|$ *as a bound for* Q'.

We will have integer variables h_X *for subsets* X *of the query variables, and we will get inequalities on* h_X *based on the atoms containing variables in* X. *The rule that* $mcid$ *is a key in* mc *gives us the degree bound inequality* $h_{\{mcid,mid\}} \leq h_{\{mcid\}}$. *Similarly the fact that* mid *is a key in* t *gives* $h_{\{mid,text\}} \leq h_{\{mid\}}$. *Submodularity tells us that* $h_{\{mid,text,mcid\}} \leq h_{\{mid,text\}} + h_{\{mid,mcid\}} - h_{\{mid\}}$. *Substituting in the above and cancelling* h_{mid} *we get* $h_{\{mid,text,mcid\}} \leq h_{\{mcid\}}$.

Our stored cardinality for $|V|$ *gives us a trivial degree bound for the empty set of variables and thus using the Degree inequality and non-negativity we get* $h_{\{mcid\}} \leq |V|$. *Putting* *these together gives:*

$$h_{\{mid,text,mcid\}} \leq log(|V|)$$

and monotonicity then implies that the objective of the LP, namely $2^{h_{\{text\}}}$, is bounded by $|V|$, as required. Thus CLLP *can find a good bound on Q'. But it cannot find this upper bound on Q given the statistics and the input query alone.* □

Rewriting estimation with assumptions. The CLLP approach provides hard upper bounds, but in doing so it gives up the heuristics and independence assumptions commonly used in estimation. We now look at plugging into our semantics-aware size estimation technique approaches for estimating the size of a single rewriting that can make use of heuristics (e.g. see [Sack, 2014; Bruno and Chaudhuri, 2004]).

Our algorithm is syntax-driven, working off of a "structured" version of the QueryOf algorithm which converts a set of chase facts into a formula, preserving the compositional structure produced in our algorithm; QueryOf applied to $\mathcal{R}_1 \circ \mathcal{R}_2$ is formed by existentially quantifying the conjunction of formulas derived by stripping off quantifiers from QueryOf$(\mathcal{R}_1)$ and QueryOf$(\mathcal{R}_2)$. Our algorithm descends into the parse-tree of the structured query produced, one connective at a time. Semantics still play a role in this approach in the induction step for conjunctions. It estimates the size of $Q_1 \wedge Q_2$ by recursively estimating the size of Q_1 and Q_2 using traditional approaches (e.g. see [Sack, 2014; Bruno and Chaudhuri, 2004]), but exploiting derived rules when possible. For example, when we can derive (using rule inference) that the common variables represent a key within one of Q_1 and Q_2, we approximate via a recursive call to the other subquery. Otherwise, if the common variables do not involve a key among the subqueries of Q, then the algorithm does a recursive estimate, using standard methods for estimating a conjunction, using histograms when available (e.g. [Sack, 2014; Bruno and Chaudhuri, 2004]). For example, in estimating a conjunction of atoms in which histogram are available on both relations, indexed on positions of the common variables, independence of the distributions is assumed. We also make use of standard heuristics and assumptions in the base case; e.g. assuming uniform distribution within histogram buckets.

Example 5 *Recall the annotated rewriting $\mathcal{R}$ from Example 3. Let us see how to estimate the size of the corresponding query, using this "semantics-aware textbook approach".* QueryOf$(\mathcal{R})$ *is Q' (recall Example 1). We can derive that the common variables are a key for the left-hand side of the top-level conjunction, and we can use this to see that the size of V bounds the size of the query output.* □

Quality. With approaches that do use assumptions, we must distinguish between rewritings based on quality. The following describes different quality criteria and how we combine them.

The first measure relates to the quality of the base statistics. We use two dimensions, one the maximum of the variance over all uses of histograms and another giving the minimum of the sampling factor. In some cases, all rewritings will make use of additional independence assumptions. Our heuristic algorithm makes such assumptions whenever the common variables in a conjunction are not a key. We keep track of the number of times such assumptions are made. Another measure is the number of cases where the common variables in a conjunction form a key but not a foreign key. The algorithm uses the same approximation as in the foreign-key case, but the approximation is less tight; and thus we add a quality penalty. We consider an additional measure, relating to the "proximity" of the underlying rewriting to the input query. Following [Neumann and Moerkotte, 2011], we measure this as the size of the maximum subquery $Q' \subset Q$ that has a match into the annotations of $\mathcal{R}$–a higher score (with a max of $|Q|$) indicates higher proximity. Note that a rewriting which is a Σ-RER has the maximum possible proximity. Our QualityOf function combines these by considering quality as a vector . In Algorithm 1, we compare quality and size via a domination ordering on all dimensions. If we have multiple non-dominated vectors of quality and size, we return the median.

5 Experimental evaluation

5.1 Implementation

We implemented Algorithm 1 and the two variants for estimating a single-rewriting of Section 4 in Java. The variant using CLLP is denoted $\mathsf{SEUSS}^{\mathsf{CLLP}}$ and the approach using top-down estimation is denoted $\mathsf{SEUSS}^{\mathsf{HEUR}}$. From each histogram on a subset of the positions of relation T we extracted degree bounds for CLLP, taking the maximum size of the number of tuples in any bucket. This gives a correct degree bound assuming 100% sampling of the dataset. Although the sampling policies of the DBMS's in our experiments varied considerably, we found that CLLP still produced hard upper bounds in practice. In solving the LP produced by CLLP we used the IBM ILOG CPLEX Java library.

5.2 Tools

In order to perform the evaluation we needed to develop some infrastructure.

Generation tools. While we used existing benchmarks for schemas, we needed to develop our own generators for views and (for some scenarios) queries. Our query-generation algorithm takes the following inputs: a schema with constraints; the desired number of key-to-foreign key conjunctions $q_{|fk-joins|}$ in the output query; the desired number of non key-to-foreign key conjunctions $q_{|joins|}$ in the output query; the desired number of "filters" (in logical terms, constants in atoms) $q_{|filters|}$ in the output query; the percentage of variables $q_{|projections\%|}$ that will be free. We first iteratively create $q_{|fk-joins|}$ conjunctions, picking first a random relation R and position i from the schema and then a random relation and position that is associated via a key-to-foreign key constraint with position i of R. We employ the same procedure to create conjunctions over positions that are associated via a key-to-foreign key constraint. In this case we give as input to the generator pairs of positions of relations that produce non-empty answers when conjoined. We then proceed similarly to create "selections", first choosing a position including a variable that does not appear in a conjoined atom and then a randomly selected value from the domain of the position. We then randomly select $q_{|projections\%|}$ variables to be made free. After these steps we execute the output queries and we keep those returning non-empty results. We generate view definitions based on the same algorithm above. An additional parameter #views controls the number of views generated.

Infrastructure for comparison with commercial DBMSs. We compared the estimates produced by accuracy of our semantics-aware size estimation algorithm with that of the size estimation modules of IBM DB2 10.5 and Microsoft

SQL Server 2014 DBMSs. Both can make use of views in size estimation, including views that are *exclusively* used for estimation purposes, denoted *statistical views*.

DB2 supports a broad range of statistical views, and we defined statistical views using the command ALTER VIEW <VIEW> ENABLE QUERY OPTIMIZATION . Statistics collection on base relations and views was done by running the command RUNSTATS ON < TABLE_OR_VIEW > WITH DISTRIBUTION AND DETAILED INDEXES ALL . This collects the size of the input relation/statistical view and its columns, as well as frequency and quantile statistics for each column of the input relation/statistical view.

SQL Server makes use of a restricted type of statistical views in query size estimation called *filtered statistics*. These are statistical views over a single relation that only allow simple filtering predicates (e.g., forbidding predicates that use the LIKE operator). We created filtered statistics using command CREATE STATISTICS statistics_name ON relation (column[,...n]) WHERE < filter_predicates >. SQL Server assumes that columns of the statistical views are ordered from left to right and builds histograms only for the first column. The remaining columns are considered for the collection of cross-column correlation statistics, called densities [Sack, 2014]. Due to the limitations of filtered statistics, we employed the following steps to model statistical views in SQL Server. Consider the statistical view V defined over relations T_i, $i = 1, \ldots, N$. Consider also the set of positions $\mathcal{C}_i$ from T_i that either participate in a conjunction with another relation T_j within V or are free in V. Then, for each statistical view V and for each relation T_i we create a set of $|\mathcal{C}_i| + 1$ filtered statistics. The first $|\mathcal{C}_i|$ filtered statistics project a single column from $\mathcal{C}_i$ while keeping any selections on T_i present in V. The last one projects all columns in $\mathcal{C}_i$ in random order, again keeping any selection conditions present in V. We used the default settings of SQL Server to automate the statistics collection process, under which the DBMS collects single- and multi-column statistics for each column that is referenced in a query. The single-column statistics include sizes and frequency histograms, while the multi-column statistics include density vectors.

The statistics that we provided to CLLP, SEUSS$^{\mathsf{CLLP}}$, and SEUSS$^{\mathsf{HEUR}}$ include the size of base relations and views if applicable and size estimates and histograms for each column of a base relation or view.

Evaluating the impact of size estimates on runtime. Following the evaluation strategy of [Leis *et al.*, 2015], we adapted PostgreSQL 9.3 to allow the optimizer to utilize size estimates from an external algorithm. This allows us to assess the impact of different size estimates on PostgreSQL's query optimizer in the IMDB scenario. First, we created indices both on the primary and the foreign keys of each relation in IMDB. Second, for each query Q from the IMDB scenario, we computed all connected subqueries of it, and then estimated its size using the size estimation module of each system: (SQL Server, DB2, CLLP, SEUSS$^{\mathsf{CLLP}}$, and SEUSS$^{\mathsf{HEUR}}$). Finally, we injected the estimated sizes for the subqueries of Q into PostgreSQL and executed Q in PostgreSQL using the EXPLAIN ANALYZE command. For our experiments, we used the default settings of the built-in PostgreSQL optimizer.

The running time of SEUSS$^{\mathsf{CLLP}}$, and SEUSS$^{\mathsf{HEUR}}$ is a few seconds at most. The total time spent for query size estimation and query evaluation was thus several orders of magnitude times lower than the time spent for query execution by commercial DMBS's (see Figure 2(a)); we thus focus on accuracy of estimation in the results below.

5.3 Hardware configuration

The runtime experiments were performed on an linux machine with i7 processor and 8GB memory. The size estimation for each algorithm was done on the same machine, with the exception of the estimation with SQL Server, which was run on a Windows 7 machine, also with an i7 processor and 8GB memory. Note that we are never comparing SQL Server and DB2 runtime performance in these experiments, so the difference in platform does not impact any of our results.

5.4 Scenarios

The test scenarios for our accuracy and runtime evaluation cover a variety of data sizes, degrees of column correlation, base statistics, and constraints.

TPC-H scenarios. TPC-H [TCPH, 2017; Barata *et al.*, 2015] is a well-established benchmark containing eight relations and eight foreign key constraints. We created two scenarios using TPC-H. In the first (TPCH-MANUAL), both the queries and the views were developed manually. It comprises 13 cases, each with queries consisting of up to six conjunctions. There are at most four views per case, with each consisting of up to three conjuncts. The cases were chosen so that finding rewritings requires reasoning with foreign keys over both the base relations and the statistical views In the second scenario (TPCH-SYNTHETIC), both the queries and the statistical views were created using our generators. we defer the details to the appendix.

For both test scenarios we populated the schema with data using the TCP-H data generator with scaling-factor=1, resulting in a 1GB database. We gave our algorithm and that of the DBMS's full statistics for all base relations and statistical views.

IMDB scenario. The IMDB benchmark was designed to assess the performance of query size estimation algorithms [Leis *et al.*, 2015]. The schema consists of twenty-one relations and twenty-three foreign key constraints and is populated with 5GB of data from the Internet Movie Data Base. The benchmark comes with a set of 113 manually created queries, including a wide range of SQL features beyond conjunctive queries. We experimented on a subset of 14 queries, after removing any aggregate operators in the SELECT clause.

IMDB is a challenging benchmark for size estimation algorithms due to the nature of the data: inter-relation correlations and non-uniform data distributions make independence assumptions particularly dangerous. Similarly to scenarios TPCH-MANUAL and TPCH-SYNTHETIC, we assumed that we have full statistics for all base relations and statistical views.

LUBM scenarios. LUBM is a popular benchmark in the Semantic Web community. It is interesting for a constraint-based optimizer as it comes with a rich set of TGDs and a set of benchmark queries. Using the LUBM data generator we produced instances with $12m$ RDF triples. We converted these into relational form using a standard encoding.

Using the 14 queries provided by the benchmark, we created two different scenarios. In the first scenario (LUBM-100%) we assumed that all of the base relations have statistics, while in the second scenario (LUBM-50%), we assumed statistics for 50% of the relations in the input queries. We did not define statistical views for either of these scenarios as our objective was to assess the impact of general constraints in size estimation. For scenario LUBM-50% we collected statistics only for a subset of the relations mentioned in the query and for all relations not mentioned in the query. In DB2 it was easy to realise these scenarios: the DBMS does not collect statistics for a base relation or a statistical view unless we run the command RUNSTATS . In SQL Server we did not find a direct way to disable the statistics collection module for a subset of the relations. We simulated the situation where no statistics are available on relation T in SQL Server by rewriting the query to use a temporary relation created on-the-fly from relation T instead of directly referencing relation T.

5.5 Results

The experimental results are summarized in Figure 2. They show the runtime when injecting estimated sizes into PostgreSQL's optimizer (Figure 2(a)) and the *q-error* between the actual size and the derived estimate, defined as $\max\{\hat{s}/s, s/\hat{s}\}$, where s is the actual size and $\hat{s}$ its estimate (Figure 2(b)-(f)). This statistic captures the intuition that only relative differences matter for making planning decisions. Below, we summarize some insights:

1. the size estimates of SEUSS improve the runtime performance (see Figure 2(a))
2. SEUSS$^{\text{CLLP}}$ results both in better runtime and better size estimates than CLLP (see 2(a)-(f))
3. SEUSS$^{\text{HEUR}}$ beats SEUSS$^{\text{CLLP}}$ and commercial database management systems in terms of q-error (see Figures 2(a)-(f))
4. the benefits in terms of q-error of SEUSS$^{\text{HEUR}}$ become greater in the absence of statistics for subsets of the relations (see Figure 2(f))

The explanation for the first insight is that major size misestimations of other systems lead PostgreSQL's optimizer towards choosing the wrong implementations of operators (e.g., they make use of a naïve nested-loop join instead of hash join). Underlying the second insight is that SEUSS$^{\text{CLLP}}$, in contrast to CLLP, makes use of inferred rules. An example is query Q_{10c} from IMDB:

$$Q_{10c}(prid, id, cid, ctid, id', rid, mid) = \\ chn(prid) \wedge ci(id, mid, rid, prid, \text{producer}) \wedge cn(cid, \text{us}) \wedge \\ ct(ctid) \wedge mc(id', mid, cid, ctid) \wedge rt(rid) \wedge t(mid, 1990)$$

In the above producer, us and 1990 are constants. The actual size of Q_{10c}'s output on the dataset is 10. Our underlying statistics reveal that the view defined by

$$V_{10c}(prid, id, id', rid, mid) \leftrightarrow \\ chn(prid) \wedge ci(id, mid, rid, prid, \text{producer}) \wedge \\ mc(id', mid, cid, ctid) \wedge rt(rid) \wedge t(mid, 1990)$$

has 52 tuples. Applying CLLP to the query with full statistics yields a vast overestimate of the size, 78782313696. In contrast, SEUSS$^{\text{CLLP}}$ (and SEUSS$^{\text{HEUR}}$) identifies that the rewriting $mc(id', mid, cid, ctid) \wedge V_{10c}(prid, id, id', rid, mid)$ provides an exact upper bound of 52 for Q_{10c}, exploiting the key-to-foreign-key dependency between mc and V_{10c}. Underlying the third insight is the fact that commercial DBMSs make limit use of views, and instead assume that the selectivity of constants in a query is independent of that of conjuncts. In our running example query, having actual output size 5479, where we have statistics on the view

$$V_{2a}(text) \leftrightarrow \\ mc(mcid, cid, mid) \wedge k(kid, \text{c}) \wedge \\ mk(mid, kid) \wedge t(mid, text)$$

Even with access to these statistics, DB2 has a q-error of over 1000. Instead of rewriting to use V_{2a} it makes several assumptions of independence (see Example 1). SEUSS$^{\text{HEUR}}$, using only(!) statistics on view V_{2a} returns an estimate with a q-error below 100.

To understand the fourth point, consider query Q5 from LUBM, asking for people who are members of a particular university u_0, expressed in logic as

$$Person(pid) \wedge memberOf(pid, u_0)$$

In LUBM-50%, even without having any statistics on relation $memberOf$, SEUSS$^{\text{HEUR}}$ returns a size estimate of 742, close to the actual query size of 719. SEUSS$^{\text{HEUR}}$ achieves this by taking advantage of the foreign key dependency between $memberOf$ and $member$, which does have statistics. Commercial DBMSs give estimates that are essentially random for these examples, since they have no principled way of compensating for the lack of relevant statistics.

6 Conclusion

Estimating the number of answers to a query is a core challenge for data and knowledge management. While certainly many techniques and tools are required, in this paper we show that the more extensive use of semantic information can play an important role. In the process we have given the first experimental study of the use of semantic information in size estimation, measuring both the impact on accuracy and the final "bottom line impact" on query evaluation time.

We have focused here on rules extractable from SQL datasources, in order to make our experimental comparison with commercial systems valid. But we believe that the role of semantics-based rewriting in estimation will become even greater in data integration settings, where even richer semantics are present.

References

[Atserias *et al.*, 2013] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. *SIAM J. Comp.*, 42(4), 2013.

[Barata *et al.*, 2015] Melyssa Barata, Jorge Bernardino, and Pedro Furtado. An overview of decision support benchmarks: Tpc-ds, TPC-H and SSB. In *WorldCIIST*, pages 619–628, 2015.

[Beeri and Vardi, 1984] Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.

[Bruno and Chaudhuri, 2002] N. Bruno and S. Chaudhuri. Exploiting statistics on query expressions for optimization. In *SIGMOD*, 2002.

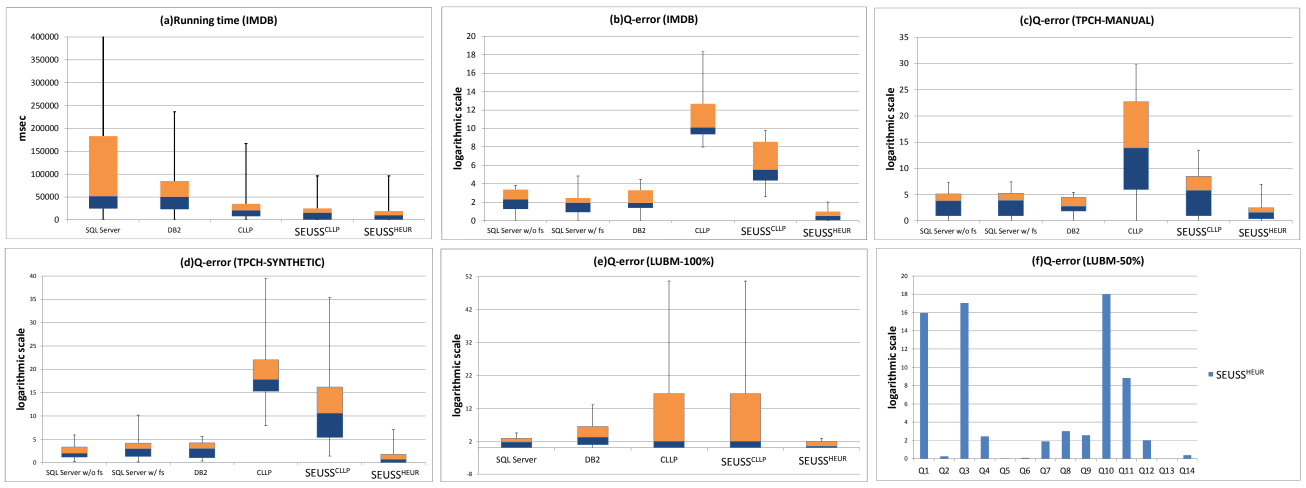

Figure 2: Runtime and q-error results

[Bruno and Chaudhuri, 2004] N. Bruno and S. Chaudhuri. Conditional selectivity for statistics on query expressions. In *SIGMOD*, 2004.

[Calvanese *et al.*, 2008] Diego Calvanese, Evgeny Kharlamov, Werner Nutt, and Camilo Thorne. Aggregate queries over ontologies. In *ONISW*, 2008.

[Christodoulakis, 1984] S. Christodoulakis. Implications of certain assumptions in database performance evaluation. *TODS*, 9(2):163–186, 1984.

[Deutsch *et al.*, 2006] A. Deutsch, L. Popa, and V. Tannen. Query reformulation with constraints. *SIGMOD Record*, 35(1), 2006.

[Fagin *et al.*, 2005] R. Fagin, P.G. Kolaitis, R.J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[Gottlob *et al.*, 2012] G. Gottlob, S. T. Lee, G. Valiant, and P. Valiant. Size and treewidth bounds for conjunctive queries. *JACM*, 2012.

[Ioannidis and Christodoulakis, 1991] Y. Ioannidis and S. Christodoulakis. On the propagation of errors in the size of join results. In *SIGMOD*, 1991.

[Ioannidis, 2003] Y. Ioannidis. The history of histograms (abridged). In *VLDB*, 2003.

[Joglekar and Ré, 2016] M. Joglekar and C. Ré. It's all a matter of degree: Using degree information to optimize multiway joins. In *ICDT*, 2016.

[Khamis *et al.*, 2016] M. Khamis, H. Ngo, and D. Suciu. Computing join queries with functional dependencies. In *PODS*, 2016.

[Kostylev and Reutter, 2015] Egor V. Kostylev and Juan L. Reutter. Complexity of answering counting aggregate queries over DL-Lite. *J. Web Sem.*, 33:94–111, 2015.

[Leis *et al.*, 2015] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *VLDB*, 9(3), 2015.

[Maier *et al.*, 1979] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *TODS*, 4(4), 1979.

[Neumann and Moerkotte, 2011] T. Neumann and G. Moerkotte. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *ICDE*, 2011.

[Ngo *et al.*, 2014] H. Q. Ngo, C. Ré, and A. Rudra. Skew strikes back: New developments in the theory of join algorithms. *SIGMOD Record*, 42(4):5–16, 2014.

[Popa, 2000] L. Popa. *Object/Relational Query Optimization with Chase and Backchase*. PhD thesis, U. Penn., 2000.

[Pratt-Hartmann, 2009] Ian Pratt-Hartmann. Data-complexity of the two-variable fragment with counting quantifiers. *Inf. Comput.*, 207(8):867–888, 2009.

[Reddy and Haritsa, 2005] N. Reddy and J. R. Haritsa. Analyzing plan diagrams of database query optimizers. In *VLDB*, 2005.

[Sack, 2014] J. Sack. Optimizing your query plans with the SQL Server 2014 cardinality estimator. Microsoft white pages, April 2014.

[TCPH, 2017] Transaction processing performance council, 2017. *TPC-H Benchmark*. `http://www.tpc.org`.

[Toman and Weddell, 2011] D. Toman and G. Weddell. *Fundamentals of Physical Design and Query Compilation*. Morgan Claypool, 2011.