On Scale Independence for Querying Big Data

On Scale Independence for Querying Big Data

Wenfei Fan
University of Edinburgh
& Beihang University
wenfei@inf.ed.ac.uk

Floris Geerts
University of Antwerp
floris.geerts@uantwerpen.be

Leonid Libkin
University of Edinburgh
libkin@inf.ed.ac.uk

ABSTRACT

To make query answering feasible in big datasets, practitioners have been looking into the notion of scale independence of queries. Intuitively, such queries require only a relatively small subset of the data, whose size is determined by the query and access methods rather than the size of the dataset itself. This paper aims to formalize this notion and study its properties. We start by defining what it means to be scale-independent, and provide matching upper and lower bounds for checking scale independence, for queries in various languages, and for combined and data complexity. Since the complexity turns out to be rather high, and since scale-independent queries cannot be captured syntactically, we develop sufficient conditions for scale independence. We formulate them based on access schemas, which combine indexing and constraints together with bounds on the sizes of retrieved data sets. We then study two variations of scale-independent query answering, inspired by existing practical systems. One concerns incremental query answering: we check when query answers can be maintained in response to updates scale-independently. The other explores scale-independent query rewriting using views.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design – *Data Models*; H.2.1 [**Database Management**]: Systems – *Query Processing*
General Terms: Theory, Languages, Algorithms
Keywords: Scale independence; big data; query answering

1. INTRODUCTION

Big data introduces challenges to the scalability of query answering. Given a query Q and a dataset D, it is often prohibitively costly to compute the answers $Q(D)$ to Q in D when D is big, *e.g.,* of PetaByte (10^{15} bytes) or ExaByte (10^{18}) size. To this end, one may want to use heuristics, "quick and dirty" algorithms which return approximate answers. However, in many applications it is a must to find exact query answers.

To cope with these, practitioners have been studying scale independence (*e.g.,* [4–6]). A query Q is said to be *scale-independent* in a dataset D *w.r.t.* M if $Q(D)$ can be computed by accessing a set D_Q of at most M tuples in D, *independent of* the size of the underlying dataset D. Here M is a non-negative integer, indicating the capacity of our available resources, such as time and space.

The need for scale independence is evident in practice. It allows us to answer Q in big D within our available resources. Moreover, if Q is scale-independent in all datasets, we can answer Q without performance degradation when D grows, *i.e.,* make Q scalable.

Scale independence per se is not easy to achieve, nor is it easy to test for, as we shall show. Nonetheless, there are many practical scenarios where scale independence is achievable, roughly classified into three groups below.

(1) Additional information about accessing information in a dataset, which is typically provided in the form of indices and/or cardinality constraints, can make rather expressive classes of queries scale-independent. Such access information is in fact commonly available in many large datasets being used in the real world.

(2) Even if a query Q is not scale-independent, we may still make it feasible to query big data *incrementally*, *i.e.,* to evaluate Q incrementally in response to changes ΔD to D, by accessing an M-fraction of the dataset D. That is, we compute $Q(D)$, once and offline, as precomputation, and then incrementally answer Q on demand.

(3) Additionally, we can sometimes achieve *scale independence using views*, *i.e.,* when a set $\mathcal{V}$ of views is defined, we rewrite Q into Q' using $\mathcal{V}$, such that for any dataset D, we can compute $Q(D)$ by using Q', which accesses materialized views $\mathcal{V}(D)$ and only a bounded amount of data from D (this is subject to the storage and maintenance costs of $\mathcal{V}(D)$).

We now illustrate these three scenarios by examples.

Example 1.1: Some real-life queries are actually scale-independent. For example, below are (slightly modified) queries taken from Graph Search of Facebook [11].

(a) Query Q_1 is to find all friends of a person p who live in NYC, from a dataset D_1. Here D_1 consists of two relations specified by $\mathsf{person}(\mathsf{id}, \mathsf{name}, \mathsf{city})$ and $\mathsf{friend}(\mathsf{id}_1, \mathsf{id}_2)$, recording the basic information of people (with a key id) and their friends, respectively. Query Q_1 can be written as follows:

$$Q_1(\mathsf{p}, \mathsf{name}) = \exists \mathsf{id}\big(\mathsf{friend}(\mathsf{p}, \mathsf{id}) \wedge \mathsf{person}(\mathsf{id}, \mathsf{name}, \mathrm{NYC})\big).$$

In personalized social searches we do not want to compute the entire answer to Q_1, but rather do it for a specified

person p_0; that is, given p_0, we want to find all values of name so that (p_0, name) is in the answer to Q_1.

The dataset D_1 is often big in real life: Facebook has more than 1 billion users with 140 billion friend links [10]. A naive computation of the answer to Q_1, even if p_0 is known, may fetch the entire D_1, and is cost prohibitive.

Nonetheless, we can compute $Q_1(p_0, D_1)$ by accessing only a small subset D_{Q_1} of D_1. Indeed, Facebook has a limit of 5000 friends per user (cf. [5]), and id is a key of person. Thus by using indices on id attributes, we can identify D_{Q_1}, which consists of a subset D_f of friend including all friends of p_0, and a set D_p of person tuples t such that $t[\mathsf{id}] = t'[\mathsf{id2}]$ for some tuple t' in D_f. Then $Q_1(p_0, D_{Q_1}) = Q_1(p_0, D_1)$. Moreover, D_{Q_1} contains at most 10000 tuples of D_1, and is much smaller than D_1. Thus Q_1 is scale-independent in D_1 *w.r.t.* $M \geq 10000$.

This illustrates the key ingredients of the recipe for scale independence. First, we may have to fix values of some parameters of the query (p_0 for p above). Second, we may need access information telling us that based on some key values, tuples can be fetched efficiently, and there is an upper limit on the number of tuples fetched.

(b) Query Q_2 is to find from D_2 all restaurants rated A in NYC, where p's friends in NYC have been. Dataset D_2 consists of four relations: person and friend as above, and relations specified by $\mathsf{restr}(\mathsf{rid}, \mathsf{name}, \mathsf{city}, \mathsf{rating})$ (with rid as a key) and $\mathsf{visit}(\mathsf{id}, \mathsf{rid})$. Here Q_2 is

$$Q_2(\mathsf{p}, \mathsf{rn}) = \exists \mathsf{id}, \mathsf{rid}, \mathsf{pn}\big(\mathsf{friend}(\mathsf{p}, \mathsf{id}) \wedge \mathsf{visit}(\mathsf{id}, \mathsf{rid}) \wedge \mathsf{person}(\mathsf{id}, \mathsf{pn}, \mathrm{NYC}) \wedge \mathsf{restr}(\mathsf{rid}, \mathsf{rn}, \mathrm{NYC}, A)\big).$$

Again, we want to find the answers for a given person p_0. However, unlike D_1, dataset D_2 imposes no restriction on the number of restaurants in NYC or on the number of restaurants which a person visits.

Nonetheless, in this case we can *incrementally* evaluate Q_2 scale-independently, leveraging the old output $Q_2(p_0, D_2)$. Given a set ΔD_2 of insertions to visit, one can compute $Q_2(D_2 \cup \Delta D_2)$ by fetching at most $3|\Delta D_2|$ tuples from D_2, where $|S|$ denotes the number of tuples in S. Indeed, for each tuple $(\mathsf{id}, \mathsf{rid})$ in ΔD_2, we fetch the restaurant identified by the key rid, the friend of p_0 identified by id, and person tuple (identified by id) to verify that the friend lives in NYC, via indexing. Let S be the set of the names of the restaurants that are fetched and visited by NYC friends. Then $S \cup Q_2(D_2) = Q_2(D_2 \cup \Delta D_2)$. Note that ΔD_2 is often small in practice. If $3|\Delta D_2| \leq M$, then the *incremental evaluation* of Q_2 is scale-independent *w.r.t.* M.

(c) Assume that two views are defined: V_1 contains all restaurants in NYC, and V_2 is a subset of $\mathsf{visit}(\mathsf{id}, \mathsf{rid})$ such that id lives in NYC (restaurants visited by NYC locals). Using V_1 and V_2, query Q_2 can be rewritten as:

$$Q_2'(\mathsf{p}, \mathsf{rn}) = \exists \mathsf{id}, \mathsf{rid}\big(\mathsf{friend}(\mathsf{p}, \mathsf{id}) \wedge V_2(\mathsf{id}, \mathsf{rid}) \wedge V_1(\mathsf{rid}, \mathsf{rn}, A)\big).$$

Then for all datasets D_2, by using materialized views $V_1(D_2)$ and $V_2(D_2)$, Q_2' needs to fetch at most 5000 friend tuples from D_2 for $\mathsf{p} = p_0$, *i.e.,* for a fixed p, Q_2 is scale-independent in all D_2 by using V_1 and V_2. Here we assume that the views are cached in memory and can be efficiently retrieved. That is, for scale independence using views to be effective, the materialized views should be of small size. □

All three examples thus show that it is feasible to answer a query Q in a big dataset D by accessing a bounded amount of data. To make practical use of scale independence, however, several questions have to be answered. Given Q and D, can we decide whether Q is scale-independent in D? If such an identification is expensive, can we find sufficient conditions for scale independence, perhaps using indices and other access information? If Q is determined to be scale-independent in D, can we effectively identify a small $D_Q \subseteq D$ such that $Q(D) = Q(D_Q)$ and $|D_Q| \leq M$, by using available indices? And can we achieve reasonable time bounds for finding this set and for evaluating the query over it? Similar questions also arise for incremental scale independence and scale independence using views.

Contributions. Our goal is to give a formal definition of scale independence (this has not been previously done) and study the above questions along the following three lines.

Complexity of scale independence. We look at the problem of deciding, given a query Q, a dataset D, and a bound M, whether Q is scale-independent in D *w.r.t.* M, *i.e.,* $Q(D)$ can be computed by accessing at most M tuples in D. We call it QDSI (with QD emphasizing that both Q and D are inputs). We establish both combined and data complexity bounds for QDSI, when Q is a conjunctive query (CQ), a union of conjunctive queries (UCQ), or in first-order logic (FO). We also study a special case when M is a constant, *i.e.,* when the capacity of our resources is fixed. Moreover, we study another problem, QSI, to check whether Q is scale-independent in *all instances* D of a relational schema.

Most of the results are negative: for FO queries, QSI is undecidable, and few sensible CQ queries are scale-independent in all instances, while the complexity of QDSI also tends to be rather high. This naturally brings us to the next theme of our investigation.

Sufficient conditions for scale independence. Not only is the complexity of QDSI and QSI rather high, it is also impossible to capture scale-independent queries syntactically. Thus, we need to find sufficient conditions for scale independence, with additional access information.

Such additional information comes in the form of *access schemas* $\mathcal{A}$ that specify what parts of data can be efficiently retrieved from D by using indices, as practiced in real life, and in addition give cardinality restrictions on such retrieved sets of tuples. We then provide a sufficient syntactic condition for an FO query to be scale-independent under $\mathcal{A}$. The class of queries we define is compositional: it is given by a set of rules forming new queries from existing ones, and we show that each rule is optimal, *i.e.,* it cannot get any tighter. We then show that the syntactic class of queries guarantees scale independence under $\mathcal{A}$. For instance, the query Q_1 we have seen above is an example of such a query: our conditions say that when the person is fixed, and if the Facebook restriction on the number of friends applies, then the query can be executed scale-independently.

Furthermore, we introduce *embedded* access schemas to incorporate constraints commonly found in practice, such as functional dependencies. We show that some queries that are not scale-independent may become so under embedded $\mathcal{A}$ with simple constraints.

Two variants of scale independence. Finally, we extend our study to incremental scale independence and scale independence using views, which have been used in practice [6]. We investigate the following problems.

(a) ΔQSI is to decide whether *for all small updates* ΔD to D, the answer to Q on the updated database can be incrementally computed from $Q(D)$ by using at most M additional tuples from base relations in D.

(b) VQSI is to decide whether a query Q can be rewritten to another query Q' using a set $\mathcal{V}$ of views, such that *for all datasets* D, $Q(D)$ can be computed by using Q', which accesses materialized views $\mathcal{V}(D)$ and at most M tuples in the data source D.

We provide complexity bounds for these problems, and sufficient conditions for queries to be incrementally scale-independent and scale-independent using views.

To the best of our knowledge, this work is the first effort to give a formal treatment of scale independence, a notion recently proposed and implemented [4–6]. Our results provide a comprehensive picture of complexity bounds for the problem, help us identify a bounded amount of data from a large dataset for query evaluation, and suggest what indices to build on our datasets. The lower bounds also justify the adoption of approximate query answering.

Related work. The notion of scale independence was proposed in [5], to guarantee that a bounded amount of work (key/value store operations) is required to execute all queries in an application, regardless of the size of the underlying data. An extension to SQL was developed in [4] to enforce scale independence, which allows users to specify bounds on the amount of data accessed and the size of intermediate results; when the data required exceeds the bounds, only top-k items are retrieved to meet the bounds. View selection and maintenance were studied in [6], such that a bounded amount of work is needed to answer queries by query rewriting using views and materialized (precomputed) views.

The goal of this work is to give a precise notion of scale independence and study its properties. We identify problems fundamental to scale independence, provide matching complexity bounds, propose access schemas to formulate data access via indexing and constraints, and give sufficient conditions for scale independence (incrementally, or using views). The results tell us what is doable and what is not. To the best of our knowledge, no prior work has studied these.

Related to our notion of access schemas is the notion of access patterns. Access patterns require that a relation can only be accessed by providing certain combinations of attribute values. Query processing under limited access patterns has been extensively studied, *e.g.,* [7, 9, 22, 24]. In contrast to the prior work, we use access schemas to combine indexing and the amount of data retrieved, and embed cardinality constraints in an access schema. Our goal is to provide a sufficient condition for identifying what queries are scale-independent with indices and constraints, rather than to study the complexity or executable plans for answering queries under access patterns [7, 9, 22, 24].

There has been a host of work on incremental query answering (surveyed in [15]) and query rewriting using views (surveyed in [16, 19]). The prior work has mostly focused on improving performance by making maximum use of precomputed query answers or views. In contrast, incremental scale independence and scale independence using views aim to *access a minimum amount of data* in data sources, to cope with the sheer volume of big data. There has also been prior work on bounded incremental computation [28], self-maintainable views [27], queries independent of updates [21] and view complements [25], which also access limited source data or no source data at all. We will clarify the difference between those previous works and ours in Sections 5 and 6.

Related to problem QDSI is the relatively complete database problem (RCDP) studied in [12]. Given a query Q, a database D, master data D_m, a set Σ of containment constraints on D and D_m, RCDP is to decide whether D has complete information to answer Q relative to D_m and Σ, *i.e.,* for all extensions D' of D, if D' and D_m satisfy Σ, then $Q(D) = Q(D')$. In contrast, QDSI is to decide whether there exists a $D_Q \subseteq D$ such that $Q(D) = Q(D_Q)$ and $|D_Q|$ is below a bound M. The two problems are quite different, from complexity bounds to proofs. For instance, when Q is in CQ and constraints in Σ are expressed in CQ, RCDP is NEXPTIME-complete, while QDSI is Σ_3^p-complete.

There has also been recent work on querying big data, *e.g.,* on the communication complexity of parallel query evaluation [17, 18], the complexity of query processing in terms of MapReduce rounds [2, 30], and the study of query classes that are tractable on big data [13]. In contrast, this work studies whether it is feasible to compute query answers in big data by accessing a small subset of the data, and if so, how to efficiently identify this subset.

Organization. Section 2 presents notations. Section 3 establishes the complexity bounds for QDSI and QSI, and Section 4 deals with conditions for scale independence under access schemas. Sections 5 and 6 investigate ΔQSI and VQSI, respectively. Conclusions are in Section 7.

2. PRELIMINARIES

A relational schema $\mathcal{R}$ consists of a collection of relation names $(R_1, \ldots, R_n)$, with each R_i having a fixed set of attributes. We assume a countably infinite set U from which elements populating databases are drawn. That is, an instance D of $\mathcal{R}$ associates with each $R \in \mathcal{R}$ having m attributes an m-ary relation R^D over U, *i.e.,* a subset of U^m. When there is no confusion, we omit the superscript D. The set of all elements of U present in relations in D is called the *active domain* of D and is denoted by $adom(D)$.

We shall use logical languages for expressing queries declaratively. The languages $\mathcal{L}$ used here are standard relational languages (see [1] for details). We list them now, together with their relational algebra equivalents.

- Conjunctive queries (CQ) are built up from relation atoms $R_i(\bar{x})$ (for $R_i \in \mathcal{R}$), and equality atoms $x = y$ or $x = c$ (for constant c), by closing them under conjunction $\wedge$ and existential quantifier $\exists$ (*i.e.,* the class SPJ of select-project-join queries);
- Unions of conjunctive queries (UCQ) are queries of the form $Q_1 \cup \cdots \cup Q_k$, where each Q_i is in CQ for $i \in [1, k]$ (equivalently, SPJU queries);
- First-order logic queries (FO) are built from atomic formulas by using $\wedge$, $\vee$, negation $\neg$, and quantifiers $\exists$ and $\forall$ (equivalently, the full relational algebra).

If $\bar{x}$ is the tuple of free variables of Q, we shall also write $Q(\bar{x})$. Given a query $Q(\bar{x})$ with $|\bar{x}| = m$ and a database D, the answer to Q in D, denoted by $Q(D)$, is the set of tuples $\{\bar{a} \in adom(D)^m \mid D \models Q(\bar{a})\}$.

Often we need to fix values for some free variables. For a query $Q(\bar{x}, \bar{y})$ with $|\bar{y}| = m$ and a tuple $\bar{a}$ of values for $\bar{x}$, $Q(\bar{a}, D)$ denotes $\{\bar{b} \in adom(D)^m \mid D \models Q(\bar{a}, \bar{b})\}$.

If Q is a sentence (*i.e.,* it has no free variables), we refer to it as a *Boolean query*; such a query returns true or false. To distinguish queries that have free variables, we shall call them *data selecting* queries; for such a query Q, the answer $Q(D)$ is a set of (nonempty) tuples.

Remark. All our complexity results for CQ also hold for UCQ. Hence, in what follows, we will only mention CQ and FO when reporting complexity bounds.

3. SCALE-INDEPENDENT QUERIES

The key idea behind scale independence is that we can find a small subset of a database D so that a query Q can be answered over that subset, rather than over the entire D. In this section we define the notion of scale independence formally, study its basic properties, and establish the complexity of problems associated with it.

Let $\mathcal{R}$ be a relational schema, D a database of this schema, Q a query in language $\mathcal{L}$, and M a non-negative integer. Let $|D|$ denote the size of D, measured as the total number of tuples in relations of D.

We say that Q is *scale-independent in* D w.r.t. M if there exists a subset $D_Q \subseteq D$ such that

- $|D_Q| \leq M$ and
- $Q(D_Q) = Q(D)$.

That is, to answer Q in D, we need only to fetch at most M tuples from D, *regardless of how big* D *is*.

We refer to D_Q as a *witness* for scale independence of Q in D *w.r.t.* M. We write $\mathsf{SQ}_{\mathcal{L}}(D, M)$ for the set of all $\mathcal{L}$ queries that are scale-independent in D *w.r.t.* M.

We say that Q over schema $\mathcal{R}$ is *scale-independent* w.r.t. M if Q is scale-independent in D *w.r.t.* M for *all databases* D of $\mathcal{R}$, and write $\mathsf{SQ}_{\mathcal{L},\mathcal{R}}(M)$ for the set of all $\mathcal{L}$ queries Q that are scale-independent *w.r.t.* M.

For instance, for $Q_1(\mathsf{p}, \mathsf{name})$ and D_1 given in Example 1.1, the query $Q_1(p_0, \mathsf{name})$ with a given person p_0 is in both $\mathsf{SQ}_{\mathcal{L}}(D_1, 10000)$ and $\mathsf{SQ}_{\mathcal{L},\mathcal{R}}(10000)$, under the constraints that limit 5000 friends per person and id is a key of person.

These two notions lead to two problems of determining scale independence, *i.e.,* whether there exists a witness at all, denoted by QDSI and QSI. They are stated as follows.

- Problem $\mathsf{QDSI}(\mathcal{L})$:
  - INPUT: A relational schema $\mathcal{R}$, an instance D of $\mathcal{R}$, a query $Q \in \mathcal{L}$ over $\mathcal{R}$, and $M \geq 0$.
  - QUESTION: Is Q in $\mathsf{SQ}_{\mathcal{L}}(D, M)$?
- Problem $\mathsf{QSI}(\mathcal{L})$:
  - INPUT: A schema $\mathcal{R}$, a query $Q \in \mathcal{L}$ over $\mathcal{R}$, and $M \geq 0$.
  - QUESTION: Is Q in $\mathsf{SQ}_{\mathcal{L},\mathcal{R}}(M)$?

Problem QDSI tests *query-database* scale independence. It is highly relevant in practice since one often wants to know whether a query is scale-independent for the database at hand. The need for this is particularly evident for *e.g.,* Facebook: it maintains a single dataset D for its social data (when D is updated, we only need to consider incremental scale independence). Problem QSI is "stronger": it tests *query* scale independence for *all instances* of a schema.

For problem $\mathsf{QDSI}(\mathcal{L})$ we also have two versions to study: *data complexity*, when schema and query are fixed, but database and M may vary; and *combined complexity*, when everything $(\mathcal{R}, Q, D, M)$ is a parameter.

We now study the complexity of these problems. *All the lower bounds of this paper also hold when schema* $\mathcal{R}$ *is fixed.* We also study the case when M is fixed too.

Query-database scale independence. We first deal with the problem QDSI in which both the query and the database are part of the input. We start with combined complexity. The first result is for data-selecting queries, and we show that the problem is necessarily in the polynomial hierarchy even for simple classes of queries, and it is in PSPACE for FO.

Theorem 3.1: For data selecting queries, the combined complexity of $\mathsf{QDSI}(\mathcal{L})$ is

- Σ_3^p-complete when $\mathcal{L}$ is CQ; and
- PSPACE-complete when $\mathcal{L}$ is FO. □

Proof sketch. (1) Upper bounds. We first consider a simpler *witness problem*. That problem asks, given Q, D, M and $D' \subseteq D$ with $|D'| \leq M$, whether $Q(D) = Q(D')$, *i.e.,* it checks whether a given D' witnesses scale independence. It can be verified that the witness problem is Π_2^p-complete for CQ and PSPACE-complete for FO. Observe that solving QDSI just adds an existential guess of D' on top of the witness problem, thus taking us to the third level of the polynomial hierarchy for CQ, and staying in PSPACE for FO.

(2) Lower bounds. QDSI(CQ) is verified by reduction from the $\exists^*\forall^*\exists^*$3CNF problem, which is known to be Σ_3^p-complete [29]. The latter problem is to decide, given a sentence $\varphi = \exists X \forall Y \exists Z\, \psi(X, Y, Z)$, whether φ is true, where $\psi(X, Y, Z)$ is an instance of 3SAT, *i.e.,* ψ is $C_1 \wedge \cdots \wedge C_r$, and each C_i is a disjunction of three literals (variables or negations of variables in X, Y or Z).

To show that QDSI(FO) is PSPACE-hard, we use reduction from Q3SAT, which is PSPACE-complete (cf. [26]). Given a sentence $\varphi = P_1 x_1 \ldots P_m x_m\ \psi(x_1, \ldots, x_m)$, Q3SAT is to decide whether φ is true, where P_i is either $\exists$ or $\forall$, and ψ is an instance of 3SAT. The reduction uses a Boolean FO query and a constant M. □

Boolean queries. For Boolean queries, QDSI becomes much simpler for CQ. Indeed, if $Q(D)$ is true, then $Q(D_Q)$ is true for some D_Q such that $|D_Q| \leq \|Q\|$. For a CQ Q, we measure $\|Q\|$ as the size of the tableau of Q. This follows from the standard homomorphism semantics of CQ (see [1]). For a UCQ $Q = Q_1 \cup \cdots \cup Q_k$, we define $\|Q\|$ to be $\mathsf{max}\{\|Q_i\| \mid i \in [1, k]\}$. In practice, typically $\|Q\| \ll M \ll |D|$.

Since the PSPACE lower bound of Theorem 3.1 was verified by using a Boolean FO query, in the case of FO there is no lowering of the complexity. Thus, we have:

Corollary 3.2: For Boolean queries Q, $\mathsf{QDSI}(\mathcal{L})$ is

- in constant time (if $\|Q\| \leq M$) when $\mathcal{L}$ is CQ; and
- PSPACE-complete when $\mathcal{L}$ is FO.

for the combined complexity. □

A similar analysis gives a bound for M when Q is a data-selecting CQ query. Since for each tuple $\bar{a}$ in $Q(D)$

Query languages	Data selecting		Boolean	
	combined (Th 3.1)	data (Th 3.3)	combined (Cor 3.2)	data (Th 3.3)
CQ, UCQ	Σ_3^p-complete	NP-complete	O(1)-time	O(1)-time
FO	PSPACE-complete	NP-complete	PSPACE-complete	NP-complete
	Special case: when M is a constant			
	combined (Prop 3.4)	data (Prop 3.4)	combined (Th 3.4)	data (Th 3.4)
CQ, UCQ	Π_2^p-complete	PTIME	O(1)-time	O(1)-time
FO	PSPACE-complete	NP-complete	PSPACE-complete	PTIME

Table 1: Complexity bounds for QDSI (O(1) cases hold when $\|Q\| \leq M$)

we need at most $\|Q\|$ tuples in D to witness it, for each $M \geq \min\{|D|, \ |Q(D)| \cdot \|Q\|\}$, we have $Q \in \mathsf{SQ}_{\mathcal{L}}(D, M)$.

Data complexity. Fixing query Q makes our lives easier. For data selecting queries, QDSI($\mathcal{L}$) is down to NP-complete for all the languages. This is because the data complexity of all the languages is in PTIME. For Boolean queries, the problem is easy for CQ, but remains NP-complete for full FO.

Theorem 3.3: The data complexity of QDSI($\mathcal{L}$) for data selecting queries is NP-complete for $\mathcal{L}$ ranging from CQ to FO. For Boolean queries, it is in O(1)-time for CQ (if $\|Q\| \leq M$) but NP-complete for FO. □

Proof sketch. (1) Upper bounds. It suffices to give an NP algorithm for checking whether $Q \in \mathsf{SQ}_{\mathcal{L}}(D, M)$ when Q is a fixed FO data-selecting query. Observe that the witness problem (see proof of Theorem 3.1) is in PTIME for FO when data complexity is concerned. Hence, the additional existential guess needed to solve QDSI brings us to NP. Boolean CQ queries inherit the O(1) bound from Corollary 3.2.

(2) Lower bounds. For both fixed data-selecting query Q in CQ and fixed Boolean Q in FO, we show that QDSI is NP-hard by reductions from the set covering problem (SCP), which is known to be NP-complete (cf. [26]). Given a finite set X, a family $F = \{C_1, \ldots, C_n\}$ of subsets of X, and a positive integer k, SCP is to decide whether there exist k subsets in F whose union is X. □

When M is fixed. When we have a fixed set of resources, the bound M is a constant. Fixing M simplifies the analysis of QDSI. The combined complexity drops one level in the polynomial hierarchy for CQ, but it remains intact for FO. Data complexity becomes tractable, even for data-selecting FO queries, since only fixed-size subsets need to be checked.

Proposition 3.4: When M is fixed, the combined complexity of QDSI($\mathcal{L}$) is

- Π_2^p-complete for data-selecting queries, and is in O(1)-time for Boolean queries if $\|Q\| \leq M$, for CQ;
- PSPACE-complete for both data-selecting queries and Boolean queries in FO.

The data complexity of QDSI($\mathcal{L}$) is the same as in Theorem 3.3 except that it becomes PTIME for FO. □

Proof sketch. (1) Upper bounds. When M is fixed, we give a Π_2^p algorithm to check whether Q is in $\mathsf{SQ}_{\mathcal{L}}(D, M)$ when Q is a data-selecting query in CQ, and a PTIME algorithm when Q is a fixed data-selecting FO query. More precisely, we have a Σ_2^p algorithm for the complement problem in which the guess of D' and the guess of a witness of $Q(D') \neq Q(D)$ are combined. The O(1) and PSPACE bounds are inherited from Corollary 3.2 and Theorem 3.1, respectively.

(2) Lower bounds. For data-selecting CQ, we show it is Π_2^p-hard by reduction from $\forall^*\exists^*$3CNF [29]. It is to decide, given a sentence $\varphi = \forall X \exists Y \, \psi(X, Y)$, whether φ is true, where $\psi(X, Y)$ is an instance of 3SAT (see the proof of Theorem 3.1 for 3SAT). For Boolean FO queries, the proof of Theorem 3.1 already verified the PSPACE-hardness by using $M = 3$. □

Table 1 summarizes the complexity results for QDSI.

Query scale independence, We now look at the problem QSI($\mathcal{L}$) that checks scale independence for *all* databases. For queries Q in CQ or UCQ, the answer is 'no' in the absence of constraints on databases, unless Q is trivial (*e.g.,* it returns a constant tuple over all databases). This is due to the monotonicity of queries: we can always add tuples to the database that generate new tuples in the answer if Q is non-trivial. For full FO, as expected, the problem is undecidable. Indeed, for $M = 0$, the problem asks whether Q or its negation is finitely valid (*i.e.,* true in every finite structure), which is undecidable (cf. [23]).

Proposition 3.5: The problem QSI is undecidable even for Boolean FO queries and every fixed M. In fact for a schema $\mathcal{R}$ and $M \geq 0$, the set $\mathsf{SQ}_{\mathsf{FO},\mathcal{R}}(M)$ is not even recursively enumerable. □

Note that $Q \in \mathsf{SQ}_{\mathcal{L}}(D, |D|)$ for every language $\mathcal{L}$ and every query Q: all this says is that Q can be answered on D itself. The question is whether the $|D|$ bound can be lowered, ideally to a constant. For general data-selecting queries this may not be doable over all databases, *e.g.,* when the queries need to look at the entire input such as those that simply return the input database. But what about Boolean queries?

We say that a Boolean query Q *does not use its input fully* if there is a function $f_Q : \mathbb{N} \to \mathbb{N}$ such that $f_Q(n) < n$ for all sufficiently large n and $Q \in \mathsf{SQ}_{\mathcal{L}}(D, f_Q(|D|))$ for every D. Otherwise a query fully uses its input.

Clearly every Boolean CQ does not use its input fully: it only needs a portion of it of the size $\|Q\|$. But when it comes to FO, this is not the case. Indeed, one can easily find Boolean FO queries that fully use their input.

Proposition 3.6: There are FO Boolean queries that fully use their input. □

The results above might look negative: testing scale independence is computationally hard in the presence of data (which may be of very large size), or undecidable when we want to check whether it works on all databases (or, worse yet, the answer is simply negative). Nonetheless, this is not an atypical situation in databases, and it simply tells us that we should look for meaningful restrictions on queries to achieve scale independence. This is what we do next.

4. QUERY ANSWERING WITH ACCESS SCHEMAS

The results of the previous section indicate that without additional knowledge about the class of databases on which queries are posed, it is hard to achieve scale independence and hard to test it. We now introduce additional restrictions that will allow us to define an expressive fragment of FO admitting scale independence.

The motivation for the type of restrictions we want to use comes from access methods used in practice, and it is already implicit in Example 1.1. There are three reasons why Q_1 can be answered fast. First, Facebook imposes a limit on the number of friends. Second, for each person id, we can retrieve his/her friends (as well as other information) quickly, due to the presence of an index. And third, the query used constant person id p_0.

Thus, to be able to state that some queries are scale-independent, we need information about *access to data*: both on the speed of access, and on the amount of data that can be retrieved. We formalize this in terms of a notion of access schemas. Then we show when the combination of the syntactic shape of queries, constants used in them, and access constraints guarantees scale independence.

Access schemas and scale independence. For a relational schema $\mathcal{R} = (R_1, \ldots, R_n)$, an *access schema* $\mathcal{A}$ over $\mathcal{R}$ is a set of tuples (R, X, N, T) where

- R is a relation name in $\mathcal{R}$,
- X is a set of attributes of R, and
- $N, T \in \mathbb{N}$.

A database D *conforms to the access schema* $\mathcal{A}$ if two conditions hold for each $(R, X, N, T) \in \mathcal{A}$:

- for each tuple of values $\bar{a}$ of attributes of X, the set $\sigma_{X=\bar{a}}(R)$ has at most N tuples; and
- $\sigma_{X=\bar{a}}(R)$ can be retrieved in time at most T.

That is, one has an index on X that allows efficient retrieval of tuples from the database, and in addition there is a bound N on the number of such tuples (in the simplest case, when X is a key, the bound is 1). Moreover, the N tuples can be retrieved in T time units by using the index.

In our Facebook example, we would have a tuple $(\mathsf{friend}, \mathsf{id}_1, 5000, T)$ for some value T in the access schema, indicating that if id_1 is provided, at most 5000 tuples with such an id exist in friend, and it takes time T to retrieve those. In addition, we would have a tuple $(\mathsf{person}, \mathsf{id}, 1, T')$, saying that id is a key for person with a known time T' for retrieving the tuple for a given id.

Given a relation schema $\mathcal{R}$, an access schema $\mathcal{A}$, and a query $Q(\bar{x}, \bar{y})$, we say that Q is *$\bar{x}$-scale-independent under $\mathcal{A}$* if for each database D that conforms to $\mathcal{A}$ and each tuple $\bar{a}$ of values for $\bar{x}$, the answer $Q(\bar{a}, D)$ can be found in time that depends only on $\mathcal{A}$ and Q, but not on D. In analogy to data complexity, when the query Q is fixed but the access schema $\mathcal{A}$ may vary, we say that Q is *efficiently* $\bar{x}$-scale-independent under $\mathcal{A}$ if the time to answer $Q(\bar{a}, D)$ is polynomial in $\mathcal{A}$.

Consider again the Facebook example:
$Q_1(\mathsf{p}, \mathsf{name}) = \exists \mathsf{id}\big(\mathsf{friend}(\mathsf{p}, \mathsf{id}) \wedge \mathsf{person}(\mathsf{id}, \mathsf{name}, \mathrm{NYC})\big)$.
Then, under the access schema given above, Q_1 is p-scale-independent: for each given person p_0, the answers to Q_1 can be found in time determined by the access schema alone.

Controllability and scale independence. It is undecidable whether a query is $\bar{x}$-scale-independent under access schema $\mathcal{A}$, even if $\mathcal{A}$ is empty. Even more, the set of scale-independent queries is not recursively enumerable, as we have seen in Proposition 3.5, which rules out the existence of an effective syntax for it.

However, the lack of effective syntactic characterizations of classes of queries is common in databases. It is typically overcome by finding good and practically relevant sufficient conditions that guarantee desired properties of queries.

This is exactly what we do now: we define a syntactic class of $\bar{x}$-controlled queries for a given access schema, where $\bar{x}$ is a subset of free variables of a query, and show that each $\bar{x}$-controlled query under $\mathcal{A}$ is efficiently $\bar{x}$-scale-independent under $\mathcal{A}$. That is, an $\bar{x}$-controlled query becomes scale-independent under $\mathcal{A}$ once we fix values $\bar{a}$ for $\bar{x}$.

We now inductively define the class of *$\bar{x}$-controlled* FO queries under an access schema $\mathcal{A}$. We also say that $Q(\bar{x})$ is *controlled* if it is $\bar{x}$-controlled, *i.e.,* controlled by providing values for all its free variables. The rules for $\bar{x}$-controlled formulae are as follows:

atoms: if (R, X, N, T) is in $\mathcal{A}$, then $R(\bar{y})$ is $\bar{x}$-controlled under $\mathcal{A}$, where $\bar{x}$ is the subtuple of $\bar{y}$ corresponding to attributes in X;

conditions: if $Q(\bar{x})$ is a Boolean combination of equalities among variables in $\bar{x}$, then Q is $\bar{x}$-controlled;

disjunction: if $Q_i(\bar{y})$ is $\bar{x}_i$-controlled under $\mathcal{A}$ for $i = 1, 2$, then $Q_1(\bar{y}) \vee Q_2(\bar{y})$ is $(\bar{x}_1 \cup \bar{x}_2)$-controlled;

conjunction: if $Q_i(\bar{x}_i, \bar{y}_i)$ is $\bar{x}_i$-controlled under $\mathcal{A}$ for $i = 1, 2$, then $Q_1 \wedge Q_2$ is controlled under $\mathcal{A}$ by both $\bar{x}_1 \cup (\bar{x}_2 - \bar{y}_1)$ and $\bar{x}_2 \cup (\bar{x}_1 - \bar{y}_2)$;

safe negation: If $Q(\bar{y})$ is $\bar{x}$-controlled under $\mathcal{A}$, and $Q'(\bar{z})$ with $\bar{z} \subseteq \bar{y}$ is controlled under $\mathcal{A}$, then $Q \wedge \neg Q'$ is $\bar{x}$-controlled under $\mathcal{A}$;

existential quantification: if $Q(\bar{y})$ is $\bar{x}$-controlled under $\mathcal{A}$ and $\bar{z}$ is a subtuple of $\bar{y} - \bar{x}$, then $\exists \bar{z}\, Q$ is $\bar{x}$-controlled under $\mathcal{A}$;

universal quantification: if $Q(\bar{x}, \bar{y})$ is $\bar{x}$-controlled under $\mathcal{A}$, and $Q'(\bar{z})$ with $\bar{z} \subseteq \bar{x} \cup \bar{y}$ is controlled under $\mathcal{A}$, then $\forall \bar{y}\, \big(Q(\bar{x}, \bar{y}) \rightarrow Q'(\bar{z})\big)$ is $\bar{x}$-controlled under $\mathcal{A}$;

expansion: if $Q(\bar{y})$ is $\bar{x}$-controlled under $\mathcal{A}$ and $\bar{x} \subseteq \bar{x}' \subseteq \bar{y}$, then Q is $\bar{x}'$-controlled under $\mathcal{A}$.

Remark. We use set-theoretic operations for tuples of free variables to avoid cumbersome notations, as the meaning of those is clear from the context: for instance, $\bar{x} \cup \bar{y}$ is the tuple of all the variables used in $\bar{x}$ and $\bar{y}$, while $\bar{x} - \bar{y}$ is the subtuple of $\bar{x}$ from which variables occurring on $\bar{y}$ are eliminated.

Example 4.1: For example, under the access schema shown earlier, the query $Q_1(\mathsf{p}, \mathsf{name})$ is p-controlled. Indeed, the access schema tells us that $\mathsf{friend}(\mathsf{p}, \mathsf{id})$ is p-controlled and $\mathsf{person}(\mathsf{id}, \mathsf{name}, \mathrm{NYC})$ is id-controlled, and hence their conjunction is p-controlled. After adding an existential quantifier over id, the whole Q_1 is still p-controlled.

As another example, consider query Q_3, which revises Q_2 of Example 1.1 to find all restaurants in NYC that are rated A and were visited in a given year by p_0's friends who lived in NYC. Relation $\mathsf{visit}(\mathsf{id}, \mathsf{rid})$ is extended by including attributes $\mathsf{yy}, \mathsf{mm}, \mathsf{dd}$, indicating that person id visited restaurant rid on a given date. Then:

$$Q_3(\mathsf{rn},\mathsf{p},\mathsf{yy}) = \exists \mathsf{id},\mathsf{rid},\mathsf{pn},\mathsf{mm},\mathsf{dd}\big(\mathsf{friend}(\mathsf{p},\mathsf{id}) \wedge \mathsf{visit}(\mathsf{id},\mathsf{rid},\mathsf{yy},\mathsf{mm},\mathsf{dd}) \wedge \mathsf{person}(\mathsf{id},\mathsf{pn},\mathrm{NYC}) \wedge \mathsf{restr}(\mathsf{rid},\mathsf{rn},\mathrm{NYC},A)\big).$$

Under the same access schema $\mathcal{A}$ as before, one can derive that all base relations are only controlled by all their free variables, except for friend that is deduced by the atom rule also p-controlled, and person that is id-controlled. We have just seen that $\mathsf{friend}(\mathsf{p},\mathsf{id}) \wedge \mathsf{person}(\mathsf{id},\mathsf{pn},\mathrm{NYC})$ is p-controlled. Processing the remaining conjunctions in this way, one can see that this the subquery of Q_3 is $\{\mathsf{p},\mathsf{rid},\mathsf{yy},\mathsf{mm},\mathsf{dd},\mathsf{rn}\}$-controlled. After adding the existential quantification $\exists \mathsf{id},\mathsf{rid},\mathsf{pn},\mathsf{mm},\mathsf{dd}$, the corresponding rule tells us that Q_3 is not scale-independent. Indeed, the existential quantification "forgets" that one needs to specify values for rid, mm, dd as specified by the controlling attributes. We will see below how to enrich the access schema $\mathcal{A}$ with embedded constraints to make Q_3 scale-independent. □

We now state the main result that the syntactic condition of controllability indeed guarantees the semantic condition of scale independence.

Theorem 4.2: If an FO query Q is $\bar{x}$-controlled under an access schema $\mathcal{A}$, then it is efficiently $\bar{x}$-scale-independent under $\mathcal{A}$. □

Proof sketch. We show by induction on $Q(\bar{x},\bar{y})$ how to retrieve a set $D_Q(\bar{a}) \subseteq D$ on which $Q(\bar{a},\cdot)$ can be evaluated, *i.e.,* $Q(\bar{a},D) = Q(\bar{a},D_Q(\bar{a}))$, for a given set of values $\bar{a}$ for $\bar{x}$, and provide polynomial bounds for its size and query evaluation time. The base case is provided by the access schema. We now sketch the conjunction case; others are similar. Assume we have $Q_i(\bar{x}_i,\bar{y}_i)$ which are $\bar{x}_i$-controlled for $i = 1, 2$; also let $\bar{x}_2'$ stand for the subtuple of $\bar{x}_2$ consisting of variables that occur in $\bar{y}_1$, and $\bar{x}_2''$ for the remaining variables of $\bar{x}_2$, *i.e.,* $\bar{x}_2'' = \bar{x}_2 - \bar{y}_1$. Given values $\bar{a}_1$ and $\bar{a}_2''$ for $\bar{x}_1$ and $\bar{x}_2''$, proceed as follows. Find $D_{Q_1}(\bar{a}_1)$ by using, *e.g.,* indices. Since $Q(\bar{a}_1,D) = Q(\bar{a}_1,D_{Q_1}(\bar{a}_1))$, the number of tuples $\bar{b}$ in $Q(\bar{a}_1,D)$ is bounded too. For each such tuple $\bar{b}$, let $\bar{b}_2$ stand for the part of it corresponding to the variables in $\bar{x}_2$. Then for each tuple $(\bar{b}_2,\bar{a}_2'')$ we can find $Q_2(\bar{b}_2,\bar{a}_2'',D)$ effectively by constructing a subset $D_{Q_2}(\bar{b}_2,\bar{a}_2'')$ and evaluating the query on it; this shows that the whole conjunction (join) is scale-independent once $\bar{a}_1$ and $\bar{a}_2''$ are known. □

Intuitively, Theorem 4.2 suggests the following. First, guided by an access schema $\mathcal{A}$, we can build up indices on certain attributes of relations in an instance of schema $\mathcal{R}$. Second, capitalizing on the indices, for all instances D of $\mathcal{R}$, we can answer FO queries Q that are $\bar{x}$-controlled under $\mathcal{A}$ by retrieving a small subset $D_Q \subseteq D$, such that $Q(D) = Q(D_Q)$ and $|D_Q| \leq M$, where M can be derived from the N-values in $\mathcal{A}$. Furthermore, there exists an effective plan for identifying D_Q, which can be derived from $\mathcal{A}$ and an inductive analysis of the structure of Q. In particular, this confirms our intuition that for a fixed p_0, the query Q_1 in our example can be evaluated in a scale-independent way.

Even though the rules might look rather easy and perhaps limited in some cases (e.g., the universal quantification rule only guarantees controllability with all free variables), it is the *combination* of rules that lets us derive nontrivial controllability statements. For instance, the universal quantification rule can be used in conjunction with another query, and then it provides genuinely new information.

To give an example, assume that we have a schema with relations $R(A,B)$, $S(A,B,C)$, and $T(A,B,C)$, and suppose that (R,A,N,T) is in the access schema for some N and T. Now we are given a query

```
SELECT A, B FROM R
WHERE A=1 AND
  NOT EXISTS (SELECT * FROM S
        WHERE R.A=S.A AND R.B=S.B AND
          NOT EXISTS (SELECT * FROM T
            WHERE T.A=S.A AND T.B=S.B AND T.C=S.C))
```

What are the conditions on S and T that will make this query scale-independent? Using our rules we can quickly answer this. The query is equivalent to $R(x,y) \wedge (x = 1) \wedge \forall z\,\big(S(x,y,z) \rightarrow T(x,y,z)\big)$. Applying the conjunction and universal quantification rules, we see that it suffices for S to be (A,B)-controlled, and T to be controlled by any set of attributes (in particular, (A,B,C)-controlled) for the whole query to be scale-independent. This suggests building an index on A, B for S and an arbitrary index for T.

Optimality of the rules. One may wonder whether the rules for controllability for FO queries are optimal, *i.e.,* can they be relaxed so that they provide us with a "tighter" notion of controllability in terms of the number of values needed to guarantee scale independence? The answer is negative. Indeed, the rules – for queries of that syntactic shape – are optimal. In general, each rule is a template that applies to many queries of the same shape. Each of those templates has many instances of the form: under an access schema $\mathcal{A}$, if $Q(\bar{x})$ is $\bar{x}_1$-controlled and $Q'(\bar{y})$ is $\bar{y}_1$-controlled, then some query $Q''(\bar{z})$, built from Q and Q', is controlled by tuples $\bar{z}_1,\ldots,\bar{z}_k$ (for some of them, Q' is not needed).

We say that such a rule is *optimal* if there exists an instance of it in which the query Q'' is not controlled by any subtuple $\bar{z}$ of a minimal tuple among the $\bar{z}_i$'s. That is, in full generality, we cannot achieve smaller controlling tuples. Then the following can be easily verified.

Proposition 4.3: Each of the rules for defining the classes of controlled queries is optimal. □

Complexity of controllability. As controllability, unlike scale independence, is a purely syntactic condition, one can expect it to be decidable. In fact we pinpoint the exact complexity of it. The conjunction rule above involves two possibilities, indicating that some guessing is needed while looking for tuples controlling a query. This intuition is confirmed by NP-completeness of two problems shown below. For the second one, we say that Q is *minimally* controlled by $\bar{x}$ if it is $\bar{x}$-controlled but not $\bar{x}'$-controlled for any subtuple $\bar{x}'$ of $\bar{x}$.

- Problem QCntl:
  - Input: An access schema $\mathcal{A}$, a number $K > 0$, an FO query $Q(\bar{y})$.
  - Question: Is there $\bar{x}$ with $|\bar{x}| \leq K$ so that Q is $\bar{x}$-controlled?
- Problem $\mathsf{QCntl}_{\mathsf{min}}$:
  - Input: An access schema $\mathcal{A}$, an FO query $Q(\bar{y})$, a variable x.
  - Question: Is Q minimally controlled by some $\bar{x}$ containing x?

Theorem 4.4: The problems QCntl and $\mathsf{QCntl}_{\mathsf{min}}$ are NP-complete. They remain NP-hard for CQ. □

Proof sketch. We use reductions from problems related to candidate keys and prime attributes, cf. [1]. □

Embedded controllability and query answering under constraints. So far we have looked at access schemas, which tell us that, given values of certain attributes, there is a bound on the set of tuples having those attribute values. But we often have cases when such constraints are *embedded*, *i.e.,* they do not apply to the whole set of attributes. For instance, consider a relation $\mathsf{visit}(\mathsf{id}, \mathsf{restaurant}, \mathsf{yy}, \mathsf{dd}, \mathsf{mm})$ indicating that a person id visited a restaurant on a given date. Then we can add to the access schema information stating that for every year yy, there is a limit on the number of retrieved months (mm) and days (dd) (namely 366), and those values can be efficiently found.

Another reason to consider such embedded statements is that they make it possible to incorporate constraints such as functional dependencies (FDs) into access schemas: an FD $X \to Y$ says that once X is fixed, we have just one possibility for the values of Y.

Formally, embedded constraints in an access schema are tuples $(R, X[Y], N, T)$ with $X \subseteq Y$ being sets of attributes of R, indicating that for a given tuple $\bar{a}$ of values of X, the result of $\pi_Y(\sigma_{X=\bar{a}}(R))$ has at most N tuples and can be found in time T. Note that previous statements (R, X, N, T) are just a special case when $Y = attr(R)$, the set of all attributes in R. An FD $X \to Y$ with a time guarantee T to retrieve values of Y for given values of X is just $(R, X[X \cup Y], 1, T)$ in the access schema.

We can then extend the rules of controllability to define what it means for a query $Q(\bar{z})$ to be $\bar{x}[\bar{y}]$-controlled (under $\mathcal{A}$) when $\bar{x} \subseteq \bar{y} \subseteq \bar{z}$. Most of the rules just mimic those for controllability except two that are similar to inference rules for FDs. Below we give two sample controllability rules (rules 1,2) and the two inference rules (rules 3,4).

1. if $(R, X[Y], N, T)$ is in $\mathcal{A}$, then $R(\bar{z})$ is $\bar{x}[\bar{y}]$-controlled, where $\bar{x}$ and $\bar{y}$ are subtuples of $\bar{z}$ corresponding to attributes in X and Y;
2. if $Q_i(\bar{x}_i, \bar{z}_i)$ is $\bar{x}_i[\bar{y}_i]$-controlled $i = 1, 2$, then $Q_1 \wedge Q_2$ is $(\bar{x}_1 \cup (\bar{x}_2 - \bar{y}_1))[\bar{x}_1\bar{x}_2\bar{y}_1\bar{y}_2]$-controlled (and likewise for the symmetric case);
3. if $Q(\bar{z})$ is $\bar{x}[\bar{y}]$-controlled and $\bar{x}' \subseteq \bar{z}$, then Q is $(\bar{x} \cup \bar{x}')[\bar{y} \cup \bar{x}']$-controlled;
4. $Q(\bar{z})$ is $\bar{x}[\bar{y}]$-controlled and $\bar{x}'[\bar{y}']$-controlled for $\bar{x}' \subseteq \bar{y}$, then Q is $\bar{x}[\bar{y} \cup \bar{y}']$-controlled.

Then one adapts the proof of Theorem 4.2 to show:

Proposition 4.5: If $Q(\bar{x}, \bar{y}, \bar{z})$ is $\bar{x}[\bar{x} \cup \bar{y}]$-controlled under $\mathcal{A}$, then $\exists \bar{z}\ Q(\bar{a}, \bar{y}, \bar{z})$ is efficiently scale-independent under $\mathcal{A}$ for each $\bar{a}$. □

By Proposition 4.5, we can process some queries scale-independently under constraints, as shown below.

Example 4.6: As we have seen in Example 4.1, query Q_3 is not scale-independent, even if p and yy are fixed. In contrast, below we show that Q_3 becomes scale-independent after adding two embedded statements to the access schema. One is $(\mathsf{visit}, \mathsf{yy}[\mathsf{yy}, \mathsf{dd}, \mathsf{mm}], 366, T)$, which simply says that a year has at most 366 days; and the other is an FD $\mathsf{id}, \mathsf{yy}, \mathsf{dd}, \mathsf{mm} \to \mathsf{rid}$, saying that on a given day, each person id dines out at most once, even in NYC (we assume the FD to be effective).

Given these two embedded statements, one can apply the rules to derive that subquery visit is (id, yy)-controlled. Together with the assumption that id forms an index for person and that restr is city-controlled, this shows that query Q_3 is (p, yy)-controlled. Thus, if we want to find, for instance, all A-rated NYC restaurants that were visited by a NYC friends of p_0 in 2013, we can pose the query $Q_3(\mathsf{rn}, p_0, 2013)$, which is now scale-independent. □

5. INCREMENTAL SCALE INDEPENDENCE

While some queries Q may not be in $\mathsf{SQ}_{\mathcal{L}}(D, M)$ for a database D, they may be incrementally scale-independent in D [6]. That is, we can compute $Q(D)$ *once* as precomputation, and then incrementally evaluate Q *on demand* in response to changes to D, by accessing at most M tuples from D. Incremental scale independence allows us to answer online queries efficiently, by making maximum use of previous computation.

In this section, we first formally specify incremental scale independence, and then establish its complexity. Finally, we identify sufficient conditions for a query to be scale-independent for incremental evaluation.

Incremental scale independence. We consider updates $\boldsymbol{\Delta}D = (\Delta D, \nabla D)$ to D that consist of a list of tuples ΔD to be inserted into D and a list ∇D of tuples to be deleted. It is required that ∇D be contained in D and ΔD be disjoint from D; in particular, $\Delta D \cap \nabla D = \emptyset$. We write $D \oplus \boldsymbol{\Delta}D$ to denote the database obtained by applying $\boldsymbol{\Delta}D$ to D, *i.e.,* $(D - \nabla D) \cup \Delta D$ (where updates are applied relation-wise).

The setting of *incremental query answering* is as follows. We are given a query Q and its answer $Q(D)$ on a database D. Now, for an update $\boldsymbol{\Delta}D$, we want to compute $Q(D \oplus \boldsymbol{\Delta}D)$, *i.e.,* we want a pair of queries $\boldsymbol{\Delta}Q = (\nabla Q, \Delta Q)$ that take $\boldsymbol{\Delta}D$ and D as inputs so that

$$Q(D \oplus \boldsymbol{\Delta}D) = Q(D) \oplus \boldsymbol{\Delta}Q(\boldsymbol{\Delta}D, D).$$

We use the notation $Q(D) \oplus \boldsymbol{\Delta}Q(\boldsymbol{\Delta}D, D)$ to denote $(Q(D) - \nabla Q(\boldsymbol{\Delta}D, D)) \cup \Delta Q(\boldsymbol{\Delta}D, D)$.

As shown in [14], for FO queries Q, we can effectively find ∇Q and ΔQ in FO so that $\nabla Q(\boldsymbol{\Delta}D, D) \subseteq Q(D)$ and $\Delta Q(\boldsymbol{\Delta}D, D)$ is disjoint from $Q(D)$. Typically such queries are found by propagating changes through relational algebra expressions [15]. Such queries can often be evaluated much faster than computing $Q(D \oplus \boldsymbol{\Delta}D)$ from scratch, if $\boldsymbol{\Delta}D$ is small compared to D, as found in practice [15].

To give an example, recall Q_2 and ΔD_2 from Example 1.1, where ΔD_2 inserts tuples into visit, denoted by $\Delta\mathsf{visit}(\mathsf{id}_2, \mathsf{rid})$. From Q_2 we derive $\Delta Q_2(\mathsf{rn}, \mathsf{p})$ as:

$$\Delta Q_2(\mathsf{rn}, \mathsf{p}) = \exists \mathsf{id}_2, \mathsf{rid}\big(\mathsf{friend}(p_0, \mathsf{id}_2) \wedge \Delta\mathsf{visit}(\mathsf{id}_2, \mathsf{rid}) \wedge \mathsf{restr}(\mathsf{rid}, \mathsf{rn}, \mathrm{NYC}, A)\big).$$

Here ΔQ_2 finds tuples to be inserted into $Q_2(D_2)$, such that $Q_2(D_2 \oplus \Delta D_2) = Q_2(D_2) \cup \Delta Q_2(\Delta D_2, D_2)$.

We say that Q is *incrementally scale-independent in* D w.r.t. (M, k) if *for all updates* $\boldsymbol{\Delta}D$ to D with $|\boldsymbol{\Delta}D| \leq k$,

there exists a subset $D_Q \subseteq D$ with $|D_Q| \leq M$ such that $Q(D \oplus \boldsymbol{\Delta} D) = Q(D) \oplus \boldsymbol{\Delta} Q(\boldsymbol{\Delta} D, D_Q)$. That is, to incrementally answer Q in D in response to $\boldsymbol{\Delta} D$, we need to access no more than M tuples from D, *independent of the size of the underlying* D. We consider updates $\boldsymbol{\Delta} D$ with a bounded number of tuples since in real life updates are typically frequent but *small*, very often consisting of single tuple insertions or deletions.

We write $\Delta\mathsf{SQ}_{\mathcal{L}}(D, M, k)$ for the set of all $\mathcal{L}$ queries that are incrementally scale-independent in D *w.r.t.* (M, k). For instance, Q_2 above is in $\Delta\mathsf{SQ}_{\mathcal{L}}(D_2, M, k)$ if $M \geq 3k$, when some constraints state that only updates allowed to D_2 are insertions $\Delta\mathsf{visit}(\mathsf{id}_2, \mathsf{rid})$.

We study the *incremental scale-independence problem*, denoted by $\Delta\mathsf{QSI}(\mathcal{L})$, which is stated as follows.

- PROBLEM: $\Delta\mathsf{QSI}(\mathcal{L})$.
  - INPUT: A schema $\mathcal{R}$, a query $Q \in \mathcal{L}$ over $\mathcal{R}$, an instance D of $\mathcal{R}$, and $M, k \geq 0$.
  - QUESTION: Is Q in $\Delta\mathsf{SQ}_{\mathcal{L}}(D, M, k)$?

Complexity. We next give the complexity of $\Delta\mathsf{QSI}$. For full FO, $\Delta\mathsf{QSI}$ matches the bound of QDSI for scale-independent query answering, even in simple settings, for combined complexity, and moves one level up in the polynomial hierarchy for data complexity.

Theorem 5.1: When $\mathcal{L}$ is FO, $\Delta\mathsf{QSI}(\mathcal{L})$ is

- PSPACE-complete for combined complexity, even for Boolean queries, and even if M is fixed; and
- Π_2^p-complete for data complexity (when both queries Q and $\boldsymbol{\Delta} Q$ are fixed), and coNP-complete if in addition, M is fixed. □

Proof sketch. Since guessing is free in PSPACE, and since the combined complexity of FO is in PSPACE as well, the definition of incremental scale independence naturally translates into a PSPACE algorithm. For the PSPACE hardness, we use reduction from Q3SAT for Boolean FO by using a fixed M.

For the lower bounds for data complexity, we use reductions from the $\forall^*\exists^*$3CNF problem and the complement of 3SAT, when M is not fixed and fixed, respectively. □

We next look at the case of CQ. It turns out that we have rather high bounds.

Theorem 5.2: The complexity of $\Delta\mathsf{QSI}(\mathsf{CQ})$ is

- Π_4^p-complete, or Π_2^p-complete if M also is fixed, for combined complexity, assuming that the maintenance queries $\boldsymbol{\Delta} Q$ are CQ as well; and
- Π_2^p-complete, or coNP-complete when M is in addition fixed, for data complexity. □

Proof sketch. The upper bounds follow essentially from parsing the definitions of being incrementally scale-independent and using the tableau representation of a CQ, except the case for fixed M, which is more involved. For data complexity, the maintenance query is allowed to be in FO, which also has PTIME data complexity.

The lower bounds are verified by reductions from satisfiability problems: $\forall^*\exists^*\forall^*\exists^*$3CNF for Π_4^p, $\forall^*\exists^*$3CNF for Π_2^p, and the complement of 3SAT for coNP. The coding for the data complexity with variable M is more involved, to cope with various ΔD and ∇D. □

The condition that maintenance queries are in CQ is true for insertion-only updates; in fact in this case the maintenance query can be computed in polynomial time [14]. An example of arbitrary updates admitting CQ maintenance query is when Q is key-preserving [8], if the projection attributes of Q include a key of each occurrence of base relations that are involved in Q.

Under access schemas. We now look at incremental scale independence under access schemas. Given a relational schema $\mathcal{R}$, an access schema $\mathcal{A}$, and a query $Q(\bar{x}, \bar{y})$, we say that Q is $\bar{x}$*-incrementally scale-independent under* $\mathcal{A}$ if for each tuple $\bar{a}$ of values for $\bar{x}$, and each update $\boldsymbol{\Delta} D$, the answer to maintenance queries $\boldsymbol{\Delta} Q(\bar{a}, \boldsymbol{\Delta} D, D)$ can be found in time that depends on Q, $\mathcal{A}$ and $\boldsymbol{\Delta} D$ only, but not on D. Note that $\boldsymbol{\Delta} Q$ has the same free variables $\bar{x}, \bar{y}$ as Q, but takes as its input both D and $\boldsymbol{\Delta} D$. Efficient incremental scale independence under $\mathcal{A}$ is defined along the same lines as its counterpart given in Section 4.

From results of Section 4, we immediately obtain:

Corollary 5.3: If queries $\boldsymbol{\Delta} Q(\bar{x}, \bar{y})$ are $\bar{x}$-controlled under an access schema $\mathcal{A}$ for a query Q in FO, then Q is efficiently incremental $\bar{x}$-scale-independent under $\mathcal{A}$. □

In the literature on view maintenance and incremental recomputation, queries $\boldsymbol{\Delta} Q$ are commonly derived for relational algebra queries. They are not queries that are written by the user, but rather automatically generated by the DBMS; hence it is better to produce them in their procedural version, to avoid an extra compilation stage [15]. We now show how to achieve incremental scale independence for relational algebra queries. As an intermediate step, we also show how to achieve scale independence for relational algebra.

The idea is as follows. The analog of a tuple $\bar{x}$ of variables in $\bar{x}$-controlled queries is now a set of attributes X of an expression E of relational algebra. We then produce results on scale independence when the X attributes are fixed, *i.e.,* on scale independence of queries $\sigma_{X=\bar{a}}(E)$. For each expression E we next introduce expressions E^{∇} and E^{Δ}, and, for an access schema $\mathcal{A}$, inductively generate the set $\mathsf{RA}_{\mathcal{A}}$ of pairs (E, X), where E is an expression, or an expression annotated with Δ or ∇, and X is a set of its attributes. These will tell us whether the expression is (incrementally) scale-independent for fixed values of attributes in X.

Formally, let $attr(E)$ be the set of attributes of the output of a relational algebra expression E. We assume that all selection conditions θ in σ_θ are conjunctions of equalities and inequalities. Then, for an access schema $\mathcal{A}$, the set $\mathsf{RA}_{\mathcal{A}}$ is defined inductively as follows:

Relational algebra rules:

- if $(R, X, N, T) \in \mathcal{A}$, then $(R, X) \in \mathsf{RA}_{\mathcal{A}}$;
- if $(E, X) \in \mathsf{RA}_{\mathcal{A}}$ and $X \subseteq Y$, then $(\pi_Y(E), X) \in \mathsf{RA}_{\mathcal{A}}$;
- if $(E, X) \in \mathsf{RA}_{\mathcal{A}}$ and θ is a condition, then $(\sigma_\theta(E), X - X') \in \mathsf{RA}_{\mathcal{A}}$, where X' is the set of attributes A for which θ implies that $A = a$;
- if $(E_1, X_1), (E_2, X_2) \in \mathsf{RA}_{\mathcal{A}}$ and $attr(E_1) = attr(E_2)$, then $(E_1 \cup E_2, X_1 \cup X_2) \in \mathsf{RA}_{\mathcal{A}}$;
- if $(E_1, X_1), (E_2, attr(E_2)) \in \mathsf{RA}_{\mathcal{A}}$ and $attr(E_1) = attr(E_2)$, then $(E_1 - E_2, X_1) \in \mathsf{RA}_{\mathcal{A}}$;

- if (E_1, X_1) and (E_2, X_2) are in $\mathsf{RA}_\mathcal{A}$, then $(E_1 \bowtie E_2, X_1 \cup (X_2 - attr(E_1)) \in \mathsf{RA}_\mathcal{A}$;
- if $(E, X) \in \mathsf{RA}_\mathcal{A}$ and $X \subseteq Y \subseteq attr(X)$, then $(E, Y) \in \mathsf{RA}_\mathcal{A}$.

We now show rules for expressions E^∇ and E^Δ. For this, we use queries for propagating changes through relational expressions, and apply relational algebra rules to them. We use maintenance queries from [14] since, unlike others proposed in the literature, those guarantee that $E^\nabla \subseteq E$ and $E^\Delta \cap E = \emptyset$. To give an example, consider the propagation expression $(E_1 - E_2)^\nabla = (E_1^\nabla - E_2) \cup (E_2^\Delta \cap E_1)$. If we know that E_1^∇ is controlled by a set X of attributes, and E_2 is controlled by any set of attributes (and thus by $attr(E_2)$), then $E_1^\nabla - E_2$ is controlled by X. Likewise, if E_2^Δ is controlled by Z and E_1 is controlled by anything (and thus by $attr(E_1)$), then $E_2^\nabla \cap E_1$ is controlled by Z, and $(E_1 - E_2)^\nabla$ by $X \cup Z$. Now we list the rules for E^∇ and E^Δ.

Decrement rules:

- if $R \in \mathcal{R}$, then $(R^\nabla, \emptyset) \in \mathsf{RA}_\mathcal{A}$;
- if (E^∇, X), (E, X) and (E^Δ, X) are in $\mathsf{RA}_\mathcal{A}$, and $X \subseteq Y$, then $((\pi_Y(E))^\nabla, X) \in \mathsf{RA}_\mathcal{A}$;
- if $(E^\nabla, X) \in \mathsf{RA}_\mathcal{A}$, then $((\sigma_\theta(E))^\nabla, X) \in \mathsf{RA}_\mathcal{A}$;
- if (E_i^∇, X_i), $(E_i, attr(E_i))$, and $(E_i^\Delta, attr(E_i))$ are all in $\mathsf{RA}_\mathcal{A}$ for $i = 1, 2$, then $((E_1 \cup E_2)^\nabla, X_1 \cup X_2) \in \mathsf{RA}_\mathcal{A}$;
- if (E_1^∇, X), (E_2^Δ, Z), and $(E_i, attr(E_i))$ are all in $\mathsf{RA}_\mathcal{A}$ for $i = 1, 2$, then $((E_1 - E_2)^\nabla, X \cup Z) \in \mathsf{RA}_\mathcal{A}$;
- if (E_i^∇, X_i) and (E_i, Y_i) are all in $\mathsf{RA}_\mathcal{A}$ for $i = 1, 2$, then $((E_1 \bowtie E_2)^\nabla, X_1 \cup X_2 \cup (Y_1 - attr(E_2)) \cup (Y_2 - attr(E_1)))$ is in $\mathsf{RA}_\mathcal{A}$.

Increment rules:

- if $R \in \mathcal{R}$, then $(R^\Delta, \emptyset) \in \mathsf{RA}_\mathcal{A}$;
- if (E^Δ, X) and (E, X) are in $\mathsf{RA}_\mathcal{A}$, and $X \subseteq Y$, then $((\pi_Y(E))^\Delta, X) \in \mathsf{RA}_\mathcal{A}$;
- if $(E^\Delta, X) \in \mathsf{RA}_\mathcal{A}$, then $((\sigma_\theta(E))^\Delta, X) \in \mathsf{RA}_\mathcal{A}$;
- if (E_i^∇, X_i) and $(E_i, attr(E_i))$ are all in $\mathsf{RA}_\mathcal{A}$ for $i = 1, 2$, then $((E_1 \cup E_2)^\Delta, X_1 \cup X_2) \in \mathsf{RA}_\mathcal{A}$;
- if (E_1^Δ, X_i), (E_i^∇, Z_i), and $(E_i, attr(E_i))$ are all in $\mathsf{RA}_\mathcal{A}$ for $i = 1, 2$, then $((E_1 - E_2)^\Delta, X_1 \cup Z_2) \in \mathsf{RA}_\mathcal{A}$;
- if (E_1^Δ, X_i), $(E_i^\nabla, attr(E_i))$, and (E_i, Y_i) are all in $\mathsf{RA}_\mathcal{A}$ for $i = 1, 2$, then $((E_1 \bowtie E_2)^\Delta, X_1 \cup X_2 \cup (Y_1 - attr(E_2)) \cup (Y_2 - attr(E_1))) \in \mathsf{RA}_\mathcal{A}$.

These rules tell us when relational algebra expressions are scale-independent, both in the usual way and with respect to incremental computation. We can also construct the set D_Q for maintenance queries, by applying the standard pushing selection optimizations through the expressions.

Theorem 5.4: Assume that E is a relational algebra expression, and let $\mathcal{A}$ be an access schema. Then

- if $(E, X) \in \mathsf{RA}_\mathcal{A}$, then $\sigma_{X=\bar{a}}(E)$ is scale-independent under $\mathcal{A}$; and
- if (E^Δ, X) and (E^∇, X) are in $\mathsf{RA}_\mathcal{A}$, then $\sigma_{X=\bar{a}}(E)$ is incrementally scale-independent under $\mathcal{A}$. □

Proof sketch. The first item is by induction on relational algebra expressions, similar to the proof of Theorem 4.2. For the second item, we use change propagation expressions for relational algebra queries given in [14], and analyze them to find controlling sets of attributes. □

Leveraging the special shape of maintenance queries, we have the following result for CQ. Let $\mathcal{A}(R)$ be an access schema obtained from $\mathcal{A}$ by adding $(R, \emptyset, 1, 1)$ stating that one can obtain the entire relation R in constant time. Note that it is *easier* to answer queries scale independently under $\mathcal{A}(R)$ than under $\mathcal{A}$ alone.

Proposition 5.5: Let $Q(\bar{x}, \bar{y})$ be a CQ, and $\mathcal{A}$ an access schema. (1) If Q is $\bar{x}$-scale-independent under $\mathcal{A}(R)$, then Q is incrementally $\bar{x}$-scale-independent under $\mathcal{A}$ when updates are insertions into relation R. (2) If it is also derivable by $\mathsf{RA}_\mathcal{A}$ rules that Q is controlled by all of its attributes, then Q is incrementally $\bar{x}$-scale-independent under $\mathcal{A}$ for arbitrary updates on R. □

Example 5.6: Recall query Q_2 from Example 1.1 (also used earlier in this section). Query Q_2 was not p-controllable, but when $(\mathsf{visit}, \emptyset, 1, 1)$ is added to the access schema, ΔQ_2 becomes p-controllable, and therefore, $\Delta Q_2(\mathsf{rn}, p_0)$ can be found scale independently for a fixed p_0. □

Connections with view maintenance. We conclude with a few remarks relating our notions here with several problems from incremental view maintenance.

One is the problem of self-maintainability. A query Q is *self-maintainable* if there exists a set $\mathcal{V}$ of views such that given updates $\boldsymbol{\Delta}D$ to D, both $Q(D \oplus \boldsymbol{\Delta}D)$ and $\mathcal{V}(D \oplus \boldsymbol{\Delta}D)$ can be computed from $Q(D)$, $\mathcal{V}(D)$ and $\boldsymbol{\Delta}D$, without accessing the underlying database D [27]. This notion is stronger than incremental scale independence, but only in the presence of views that need to be maintained themselves.

Even stronger than self-maintenance is the notion of *queries independent of updates* [21]: *i.e.,* whether for all databases D and $\boldsymbol{\Delta}D$, we have $Q(D) = Q(D \oplus \boldsymbol{\Delta}D)$. In fact, [21] looks at more complex updates generated by other queries. This implies incremental scale independence even with $M = 0$, but is much more restrictive.

Finally, to analyze incremental algorithms, [28] proposes to use $|\mathsf{CHANGED}| = |\boldsymbol{\Delta}D| + |\boldsymbol{\Delta}S|$, the size of the changes in the input and output. An incremental algorithm is said to be *bounded* if its cost can be expressed as a function of only $|\mathsf{CHANGED}|$, not of $|D|$. In contrast to $\Delta\mathsf{QSI}$, bounded incremental query answering concerns the size of $\boldsymbol{\Delta}Q(\boldsymbol{\Delta}D, D)$ rather than the number of tuples in D accessed. Nonetheless, if Q is in $\Delta\mathsf{SQ}_\mathcal{L}(D, M, k)$, the cost of computing $\boldsymbol{\Delta}Q(\boldsymbol{\Delta}D, D)$ is a function of $|\boldsymbol{\Delta}D|$ and M for all $\boldsymbol{\Delta}D$ with $|\boldsymbol{\Delta}D| \leq k$.

6. SCALE INDEPENDENCE USING VIEWS

We may also approach a query Q that is not scale-independent by using views [6]. The idea is to maintain a set $\mathcal{V}(D)$ of views of a database D, and compute $Q(D)$ from $\mathcal{V}(D)$ and at most M additional tuples from D.

This section studies scale independence using views. We state the problem, provide its complexity, and give sufficient conditions for scale independence by using views.

Scale independence using views. Assume a set $\mathcal{V}$ of views defined over a schema $\mathcal{R}$, in a query language $\mathcal{L}$. Consider an $\mathcal{L}$ query Q. Let Q' be a query over $\mathcal{R}$ expanded with

relations that hold the views from $\mathcal{V}$. Then Following [20], we say Q' is *a rewriting of* Q *using* $\mathcal{V}$ if $Q(D) = Q'(D, \mathcal{V}(\mathcal{D}))$ for every database D of $\mathcal{R}$, *i.e.,* $Q(D)$ can be computed by Q' and leveraging the materialized view extents $\mathcal{V}(D)$. We consider Q' that is in the same language $\mathcal{L}$, and is *polynomially bounded*, *i.e.,* its size is bounded by a polynomial in the size $|Q|$ of query and the size $|\mathcal{V}|$ of view definitions, since queries of exponential size are not much of practical use.

For instance, Example 1.1 gives a rewriting Q_2' of Q_2 using views V_1 and V_2.

A query Q is said to be *scale-independent* w.r.t. M *using* $\mathcal{V}$ if there exists a rewriting Q' of Q using $\mathcal{V}$ such that for all instances D of $\mathcal{R}$, there is a subset $D_Q \subseteq D$ with $|D_Q| \leq M$ satisfying $Q(D) = Q'(D_Q, \mathcal{V}(D))$. That is, for *all databases* D, we can compute $Q(D)$ by using materialized views $\mathcal{V}(D)$ and by accessing at most M additional tuples from in D.

We denote by $\mathsf{VSQ}_{\mathcal{L}}(\mathcal{V}, M)$ the set of $\mathcal{L}$ queries that are scale-independent *w.r.t.* M using $\mathcal{V}$.

We investigate the *scale-independence problem using views*, denoted by $\mathsf{VQSI}(\mathcal{L})$. It is stated as follows.

- PROBLEM: $\mathsf{VQSI}(\mathcal{L})$.
  - INPUT: A schema $\mathcal{R}$, a query $Q \in \mathcal{L}$, a set of $\mathcal{V}$ of $\mathcal{L}$-definable views over $\mathcal{R}$, and $M \geq 0$.
  - QUESTION: Is Q in $\mathsf{VSQ}_{\mathcal{L}}(\mathcal{V}, M)$?

In contrast to QDSI and ΔQSI that focus on a given database D, VQSI is to decide whether a query Q is scale-independent in *all instances* D of $\mathcal{R}$ by using $\mathcal{V}$, as in the study of query rewriting using views [16, 19] (hence we do not distinguish the combined and data complexity of VQSI). As opposed to QSI, VQSI explores rewriting Q' of Q to achieve scale independence by using views, when Q is not scale-independent itself. Note that QSI is a special case of VQSI when $\mathcal{V}$ is empty.

Complexity bounds. As opposed to QDSI and ΔQSI, the problem VQSI is undecidable for FO. Moreover, the complexity bounds for VQSI are rather robust: they remain intact for both data selecting and Boolean queries; and fixing M does not simplify the analysis of VQSI.

Theorem 6.1: The problem $\mathsf{VQSI}(\mathcal{L})$ is

- NP-complete when $\mathcal{L}$ is CQ; and
- undecidable when $\mathcal{L}$ is FO,

for both data selecting and Boolean queries. The complexity remains the same when M is a constant. □

Proof sketch. We show that VQSI is NP-hard for CQ by reduction from the 3-colorability problem, which is NP-complete (cf. [26]). We verify the undecidability of VQSI for FO by reduction from the satisfiability of FO, whose undecidability can be proved by using FO queries over a single binary relation schema (cf. [23]). In light of this, we prove the lower bounds using fixed $\mathcal{R}$. Moreover, the reductions use Boolean queries and a constant M.

To show that VQSI is in NP for CQ, we need to characterize what CQ rewritings are scale-independent using views. We start with the following notations. A CQ rewriting $Q'(\bar{x})$ of a CQ $Q(\bar{x})$ using $\mathcal{V}$ is of the form:

$$Q'(\bar{x}) = \exists \bar{w}\ \big(\bigwedge_{i=1}^{p} R_i(\bar{x}_i) \wedge \bigwedge_{j=1}^{q} V_j(\bar{y}_j) \wedge \phi\big),$$

where R_j is a relation atom in $\mathcal{R}$, V_j is a view literal in $\mathcal{V}$, and ϕ is a conjunction of equality atoms. We write $Q' = \exists \bar{w}\ (Q'_b \wedge Q'_v)$, where $Q'_b = \bigwedge_{i=1}^{p} R_i(\bar{x}_i)$ and $Q'_v = \bigwedge_{j=1}^{q} V_j(\bar{y}_j)$, referred to as the *base* and *view* part of Q', respectively. We eliminate ϕ by replacing a variable x with a constant c if $x = c$ can be derived from ϕ via the transitivity of its equality atoms, and enforce $x = y$ in ϕ by using the same variable.

Variables in $\bar{x}$ are called *distinguished*. We say that a variable x in $\bar{x}$ is *constrained* in Q' if either x has been instantiated to a constant c; or it is not "connected" to a base relation in Q'_b via a chain of joins, *i.e.,* there exists no set of atoms $\{S_i(\bar{v}_i) \mid 1 \leq i \leq l\}$ in Q' so that $S_1, \ldots, S_{l-1} \in \mathcal{V}$, $S_l \in \mathcal{R}$, $x \in \bar{v}_1$, and $\bar{v}_i \cap \bar{v}_{i+1} \neq \emptyset$ for all $i < l-1$. It is called *unconstrained* otherwise.

For instance, in the rewriting $Q_2'(\mathsf{p}, \mathsf{rn})$ from Example 1.1, the variable rn is unconstrained in Q_2' since it connects to base relation friend via joins.

We then show that a data selecting CQ $Q(\bar{x})$ is scale-independent *w.r.t.* M using CQ views $\mathcal{V}$ if and only if there exists a rewriting $Q'(\bar{x}) = \exists \bar{w}\ (Q'_b \wedge Q'_v)$ of $Q(\bar{x})$ using $\mathcal{V}$ such that (a) all the distinguished variables in $\bar{x}$ are constrained in Q', and (b) $\|Q'_b\| \leq M$. For Boolean CQ, condition (b) alone suffices. The same characterization remains intact when Q, $\mathcal{V}$ and Q' are in UCQ.

Based on this, we develop an NP algorithm to check whether Q is in $\mathsf{VSQ}_{\mathcal{L}}(\mathcal{V}, M)$, for UCQ Q and $\mathcal{V}$. □

Conditions for scale independence using views. Following Section 4, we say that a query Q is *$\bar{x}$-scale-independent under an access schema $\mathcal{A}$ using $\mathcal{V}$* if there exists a rewriting Q' of Q using $\mathcal{V}$, such that for each database D that conforms to $\mathcal{A}$ and each tuple $\bar{a}$ of values for $\bar{x}$, the answer $Q'(\bar{a}, D)$ can be found in time that depends only on $\mathcal{A}$, Q and $\mathcal{V}(D)$ only, but not on $|D|$.

Consider a rewriting Q' of a CQ $Q(\bar{x})$ using $\mathcal{V}$, where $Q'(\bar{x}) = \exists \bar{w}\ (Q'_b \wedge Q'_v)$ (see the proof of Theorem 6.1). We say that $Q'(\bar{x})$ is *$\bar{y}$-controlled under $\mathcal{A}$ using $\mathcal{V}$* if its base part Q'_b is $\bar{y}$-controlled under $\mathcal{A}$, following the rules for the controllability of CQ given in Section 4. Then the characterization given in the proof of Theorem 6.1 tells us that under $\mathcal{A}$, Q is $\bar{y}$-scale-independent using $\mathcal{V}$ if $Q'(\bar{x})$ is $\bar{y}$-controlled under $\mathcal{A}$ using $\mathcal{V}$ and $\bar{y}$ includes all the unconstrained distinguished variables in $\bar{x}$.

We also give a sufficient condition for an FO query Q to be scale independence using FO views. Given an FO rewriting Q' of Q, we define the *expansion* Q'_e of Q' to be the query that unfolds Q' by substituting the definition of view V for each occurrence of $V \in \mathcal{V}$ in Q'.

From Theorem 4.2 and the proof of Theorem 6.1, one can readily get the following corollary.

Corollary 6.2: (1) For a query Q and a set $\mathcal{V}$ of views in FO, if Q has a rewriting Q' whose expansion Q'_e is $\bar{x}$-controlled under an access schema $\mathcal{A}$, then Q is $\bar{x}$-scale-independent under $\mathcal{A}$ using $\mathcal{V}$. (2) For Q and $\mathcal{V}$ in CQ, if Q has a $\bar{y}$-controlled rewriting $Q'(\bar{x})$ under $\mathcal{A}$ using $\mathcal{V}$ and if $\bar{y}$ contains all unconstrained variables of Q' in $\bar{x}$, then Q is $\bar{y}$-scale-independent of under $\mathcal{A}$ using $\mathcal{V}$. □

Example 6.3: For instance, consider again the rewriting $Q_2'(\mathsf{p}, \mathsf{rn})$ using views $\mathcal{V} = \{V_1, V_2\}$ given in Example 1.1,

Under an access schema $\mathcal{A}$ that limits 5000 friends per person, its base part $\mathsf{friend}(\mathsf{p}, \mathsf{id})$ is p-controlled using $\mathcal{V}$, and hence, so is Q_2'. As a result, Q_2 is p-scale-independent under $\mathcal{A}$ using $\mathcal{V}$. In contrast, under the same $\mathcal{A}$, Q_2 is not p-scale-independent *in the absence of* $\mathcal{V}$. □

Connections. Finally, we relate scale independence using views with prior work on the study of views.

A *complete rewriting* of Q using $\mathcal{V}$ is a rewriting of Q built up with only literals in $\mathcal{V}$ and equality predicates [20]. Obviously, if Q has a complete rewriting using $\mathcal{V}$, then Q is scale-independent using $\mathcal{V}$ with $M = 0$. That is, the former is a special case of the latter.

Given a set $\mathcal{V}$ of views over a schema $\mathcal{R}$, the study of view complements (*e.g.,* [25]) is to find a (minimal) set $\mathcal{V}_c$ of views such that for all instances D of $\mathcal{R}$, D can be reconstructed from $\mathcal{V}(D)$ and $\mathcal{V}_c(D)$. When both $\mathcal{V}$ and $\mathcal{V}_c$ are available, all queries Q is scale-independent using $\mathcal{V}$ and $\mathcal{V}_c$, with $M = 0$, without accessing D. However, it is not very practical to approach scale independence by using views and their complements: $\mathcal{V}(D)$ and $\mathcal{V}_c(D)$ are no smaller than D, and it is nontrivial to compute a minimal set $\mathcal{V}_c$ of view complements.

7. CONCLUSION

We have given a formal treatment of scale independence, incremental scale independence and scale independence using views. We have established complexity bounds of QDSI, QSI, $\Delta\mathsf{QSI}$ and VQSI for CQ, UCQ and FO, for Boolean and data selecting queries, and for combined and data complexity. We have also provided sufficient conditions for FO queries to be scale-independent under (embedded) access schemas, specifying indices and constraints. The conditions also suggest how we can efficiently identify a small subset D_Q of a database D for a query Q such that $Q(D) = Q(D_Q)$.

In the future we want to study, when Q is not scale-independent in D *w.r.t.* M, what the best performance ratio is if we approximately compute $Q(D)$ by accessing at most M tuples from D. We want to strike a balance between the accuracy of the inexact query answers and the bound on the amount of data retrieved. Another question concerns the design of access schemas. In particular, we would like to see how to optimally design access schemas for a given query workload. Furthermore, an interesting extension is to consider query languages that support grouping and aggregation. We believe that sufficient conditions for scale independence can also be developed to that setting. Finally, we want to investigate how the techniques from [3] can be used for incremental scale-independent queries.

Acknowledgments. Fan and Libkin are supported in part by EPSRC EP/J015377/1, UK. Fan is also supported by 973 Programs 2012CB316200 and 2014CB340302, NSFC 61133002, Shenzhen Peacock Program 1105100030834361 and Guangdong Innovative Research Team Program 2011D005, China.

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] F. N. Afrati and J. D. Ullman. Optimizing joins in a map-reduce environment. In *EDBT*, 2010.

[3] Y. Ahmad, O. Kennedy, C. Koch, and M. Nikolic. Dbtoaster: Higher-order delta processing for dynamic, frequently fresh views. *PVLDB*, 5(10):968–979, 2012.

[4] M. Armbrust, K. Curtis, T. Kraska, A. Fox, M. J. Franklin, and D. A. Patterson. PIQL: Success-tolerant query processing in the cloud. In *VLDB*, 2011.

[5] M. Armbrust, A. Fox, D. A. Patterson, N. Lanham, B. Trushkowsky, J. Trutna, and H. Oh. Scads: Scale-independent storage for social computing applications. In *CIDR*, 2009.

[6] M. Armbrust, E. Liang, T. Kraska, A. Fox, M. J. Franklin, and D. Patterson. Generalized scale independence through incremental precomputation. In *SIGMOD*, 2013.

[7] V. Bárány, M. Benedikt, and P. Bourhis. Access patterns and integrity constraints revisited. In *ICDT*, 2013.

[8] G. Cong, W. Fan, F. Geerts, J. Li, and J. Luo. On the complexity of view update analysis and its application to annotation propagation. *TKDE*, 24(3), 2012.

[9] A. Deutsch, B. Ludäscher, and A. Nash. Rewriting queries using views with access patterns under integrity constraints. *TCS*, 371(3), 2007.

[10] Facebook. *http://newsroom.fb.com*.

[11] Facebook. Introducing graph search. *https://en-gb.facebook.com/about/graphsearch*, 2013.

[12] W. Fan and F. Geerts. Relative information completeness. *TODS*, 35(4), 2010.

[13] W. Fan, F. Geerts, and F. Neven. Making queries tractable on big data with preprocessing. In *VLDB*, 2013.

[14] T. Griffin, L. Libkin, and H. Trickey. An improved algorithm for the incremental recomputation of active relational expressions. *TKDE*, 9(3):508–511, 1997.

[15] A. Gupta and I. Mumick. *Materialized Views*. MIT Press, 2000.

[16] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[17] J. M. Hellerstein. The declarative imperative: experiences and conjectures in distributed logic. *SIGMOD Record*, 39(1):5–19, 2010.

[18] P. Koutris and D. Suciu. Parallel evaluation of conjunctive queries. In *PODS*, 2011.

[19] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.

[20] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, 1995.

[21] A. Y. Levy and Y. Sagiv. Queries independent of updates. In *VLDB*, 1993.

[22] C. Li. Computing complete answers to queries in the presence of limited access patterns. *VLDB J.*, 12(3), 2003.

[23] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[24] A. Nash and B. Ludäscher. Processing first-order queries under limited access patterns. In *PODS*, 2004.

[25] A. Nash, L. Segoufin, and V. Vianu. Views and queries: Determinacy and rewriting. *TODS*, 35(3), 2010.

[26] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[27] D. Quass, A. Gupta, I. S. Mumick, and J. Widom. Making views self-maintainable for data warehousing. In *PDIS*, 1996.

[28] G. Ramalingam and T. Reps. On the computational complexity of dynamic graph problems. *TCS*, 158(1-2), 1996.

[29] L. J. Stockmeyer. The polynomial-time hierarchy. *TCS*, 3(1):1–22, 1976.

[30] Y. Tao, W. Lin, and X. Xiao. Minimal MapReduce algorithms. *SIGMOD*, 2013.