Querying Big Data by Accessing Small Data

Citation for published version:
Fan, W, Geerts, F, Cao, Y, Deng, T & Lu, P 2015, Querying Big Data by Accessing Small Data. in *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems.* ACM, New York, NY, USA, pp. 173-184, 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Melbourne, Victoria, Australia, 31/05/15. https://doi.org/10.1145/2745754.2745771

Digital Object Identifier (DOI):
10.1145/2745754.2745771

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems

Querying Big Data by Accessing Small Data

Wenfei Fan[1,3] Floris Geerts[2] Yang Cao[1,3] Ting Deng[3] Ping Lu[3]
[1]University of Edinburgh [2]University of Antwerp [3]Beihang University
{wenfei@inf, yang.cao@}.ed.ac.uk, floris.geerts@ua.ac.be, {dengting, luping}@buaa.edu.cn

ABSTRACT

This paper investigates the feasibility of querying big data by accessing a bounded amount of the data. We study boundedly evaluable queries under a form of access constraints, when their evaluation cost is determined by the queries and constraints only. While it is undecidable to determine whether FO queries are boundedly evaluable, we show that for several classes of FO queries, the bounded evaluability problem is decidable. We also provide characterization and effective syntax for their boundedly evaluable queries.

When a query Q is not boundedly evaluable, we study two approaches to approximately answering Q under access constraints. (1) We search for upper and lower envelopes of Q that are boundedly evaluable and warrant a constant accuracy bound. (2) We instantiate a minimum set of variables (parameters) in Q such that the specialized query is boundedly evaluable. We study problems for deciding the existence of envelopes and bounded specialized queries, and establish their complexity for various classes of FO queries.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design – *Data Models*; H.2.4 [**Database Management**]: Systems – *Query Processing*

General Terms: Theory, Languages, Algorithms

Keywords: Big data; query answering; complexity

1. INTRODUCTION

Querying big data is cost prohibitive. Indeed, a linear scan of a dataset D of PB size (10^{15} bytes) takes days using a solid state drive with a read speed of 6GB/s, and it takes years if D is of EB size (10^{18} bytes) [18].

Given a query Q and a dataset D, can we efficiently compute query answers $Q(D)$ when D is big? There has been work tackling this question [11, 12, 17]. One idea is to capitalize on a set $\mathcal{A}$ of access constraints, which are a combination of indices and cardinality constraints commonly found in practice. Under $\mathcal{A}$, we study *boundedly evaluable* queries Q, such that for all datasets D that satisfy constraints in $\mathcal{A}$, there exists $D_Q \subseteq D$ such that

- $Q(D_Q) = Q(D)$, and
- the time for identifying D_Q and hence the size $|D_Q|$ of D_Q are determined by Q and $\mathcal{A}$ only.

The need for studying bounded evaluability is evident: if Q is boundedly evaluable, then $Q(D)$ can be computed by accessing (identifying and fetching) a small D_Q by using the indices in $\mathcal{A}$, in time determined by Q and $\mathcal{A}$, *not by* the size of D, no matter how big D grows. Experimenting with real-life data, we find that a large number of queries are boundedly evaluable under a small number of simple access constraints, and that such queries can be efficiently answered in big datasets that satisfy the constraints [11, 12].

Example 1.1: On the dataset D_0 of all traffic accidents in the UK from 1979 to 2005 [1], we find that 77% of conjunctive queries (CQ, *a.k.a.* SPC) are actually boundedly evaluable under a set of 84 simple access constraints, and for such queries, our query plans take 9 seconds on average as opposed to more than 14 hours by MySQL [12].

As an example, consider a query Q_0 to find the ages of drivers who were involved in an accident in Queen's Park district on May 1, 2005. The query is defined on three (simplified) relations Accident(aid, district, date), Casualty(cid, aid, class, vid) and Vehicle(vid, driver, age), recording accidents (where and when), casualties (class and vehicle), and vehicles (including driver information such as age), respectively. Query Q_0 is a conjunctive query written as

$$\begin{aligned} Q_0(x_a) = \exists\ &\mathsf{aid, cid, class, vid, dri} \\ &\big(\mathsf{Accident}(\mathsf{aid}, \text{“Queen's Park”}, \text{“1/5/2005”}) \wedge \\ &\mathsf{Casualty}(\mathsf{cid, aid, class, vid}) \wedge \mathsf{Vehicle}(\mathsf{vid, dri}, x_a)\big). \end{aligned}$$

It is costly to compute $Q_0(D_0)$ directly: the Accident, Casualty and Vehicle relations have more than 7.5, 10 and 13.5 million tuples, respectively. Nonetheless, a closer examination of D_0 reveals the following cardinality constraints:

ψ_1: Accident (date → aid, 610)
ψ_2: Casualty (aid → vid, 192)
ψ_3: Accident (aid → (district, date), 1)
ψ_4: Vehicle (vid → (driver, age), 1)

The first two constraints state that from 1979 to 2005, at most 610 accidents happened within a single day, and each accident involved at most 192 vehicles, respectively. Constraint ψ_3 says that aid is a key for Accident; similarly for ψ_4. These constraints are discovered by simple aggregate queries on D_0. Indices can be built on D_0 based on ψ_1 such

PODS'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.
Copyright © 2015 ACM 978-1-4503-2757-2/15/05 ...$15.00.
http://dx.doi.org/10.1145/2745754.2745771.

that given a date, it returns all the ids of those accidents (at most 610) that happened on the particular day; similarly for ψ_2–ψ_4. We refer to the cardinality constraints and their indices put together as *access constraints*.

Given these access constraints, we can compute $Q_0(D_0)$ by accessing at most 234850 tuples from D_0, instead of millions. (1) We identify and fetch at most 610 aid's of Accident tuples with date = "1/5/2005", using the index built on ψ_1. (2) For each aid, we fetch its Accident tuple using the index for ψ_3. We select a set T_1 of tuples with district = "Queen's Park". (3) For each tuple $t \in T_1$, we fetch a set T_2 of at most 192 vid's from Casualty tuples with aid $= t$[aid], with the index for ψ_2. (4) For each $s \in T_2$, we find a Vehicle tuple with vid $= s$[vid], using the index for ψ_4. These tuples suffice for computing $Q_0(D_0)$, $610 + 610 \times 192 \times 2$ in total, all fetched using indices. In fact, the chances are that we need to access $610 \times 2 \times 2 = 3050$ tuples only, since accidents involved two vehicles on average. Better still, no matter how big D_0 grows, as long as D_0 satisfies ψ_1–ψ_4 (possibly with cardinality bounds mildly adjusted), $Q_0(D_0)$ can be computed by accessing a small number of tuples determined by Q_0 and the bounds in ψ_1–ψ_4 only. Thus Q_0 is *boundedly evaluable under access constraints* ψ_1–ψ_4.

This approach is also effective when querying graphs. Experimenting with real-life Web graphs of billions of nodes and edges, we find that 60% of graph pattern queries via subgraph isomorphism are boundedly evaluable under simple access constraints, and that our bounded-evaluation approach outperforms conventional subgraph isomorphism methods by *4 orders of magnitude* on average [11]. □

These experimental findings verify that the bounded evaluability analysis yields a practical approach to optimizing queries on big data. The effectiveness of the approach is particularly evident for personalized searches. For example, a typical query of Graph Search, Facebook [16] is to "find me all my friends in NYC who like cycling", which only needs data relevant to a designated person (*i.e.,* "me").

However, to make effective use of the approach, several questions have to be settled. (1) Can we decide whether a query is boundedly evaluable under given access constraints? We know that this problem is undecidable for first-order logic queries (FO) [17]. Is it decidable for practical fragments of FO? (2) When queries are not boundedly evaluable, can we "approximate" them with boundedly evaluable queries that warrant reasonable approximation bounds?

Contributions. This paper tackles these questions.

Bounded evaluability. We start with a study of *the bounded evaluability problem*, denoted by BEP. Given a query Q and a set $\mathcal{A}$ of access constraints, BEP is to decide whether Q is boundedly evaluable under $\mathcal{A}$. Intuitively, it is to determine whether it is feasible to compute exact answers to Q in big datasets D by accessing a bounded amount of data from D.

It is known that BEP is undecidable for FO [17]. Hence we study BEP for several classes of FO queries, including CQ, unions of conjunctive queries (UCQ), and positive existential FO queries ($\exists$FO$^+$; *a.k.a.* SPJU). The good news is that BEP is decidable for these practical query classes. The bad news is that BEP is EXPSPACE-complete for CQ and $\exists$FO$^+$.

The complexity of BEP suggests that we develop an *effective syntax* of boundedly evaluable queries in CQ. We show that for a given set $\mathcal{A}$ of access constraints over a relational schema $\mathcal{R}$, there exists a class of CQ queries over $\mathcal{R}$ that are *covered by* $\mathcal{A}$, such that (a) it is in PTIME to decide whether a CQ is covered by $\mathcal{A}$; (b) all CQ queries covered by $\mathcal{A}$ are boundedly evaluable under $\mathcal{A}$; and (c) every boundedly evaluable CQ Q under $\mathcal{A}$ is $\mathcal{A}$-equivalent to a CQ Q' covered by $\mathcal{A}$. Here Q is $\mathcal{A}$-equivalent to Q' if for all database instances D of $\mathcal{R}$ that satisfy $\mathcal{A}$, $Q(D) = Q'(D)$. The effective syntax tells us what makes a query in CQ boundedly evaluable, and helps us design boundedly evaluable queries. Moreover, boundedly evaluable CQ queries in practice are often covered and can be syntactically checked [12]. This provides us with a PTIME method to check the bounded evaluability of conjunctive queries, which are perceived as "the most fundamental and the most widely used queries" [22].

We also extend the notion of covered queries to UCQ and $\exists$FO$^+$, and show that covered queries also provide an effective syntax for their boundedly evaluable queries. We study *the covered query problem* (CQP), to decide whether a query is covered by $\mathcal{A}$, and hence, to help us syntactically check whether a query is boundedly evaluable. We show that CQP is in PTIME for CQ, and is Π_2^p-complete for UCQ and $\exists$FO$^+$.

Boundedly evaluable envelopes. When a query Q is not boundedly evaluable under $\mathcal{A}$, we study two approaches to approximately answering Q in big data.

One approach is by means of envelopes, following [14]. We search for two queries Q_l and Q_u in the same query language of Q, such that (a) Q_l and Q_u are boundedly evaluable under $\mathcal{A}$, (b) for all datasets D, if D satisfies $\mathcal{A}$, then $Q_l(D) \subseteq Q(D) \subseteq Q_u(D)$, and (c) $|Q(D) - Q_l(D)| \leq N_l$ and $|Q_u(D) - Q(D)| \leq N_u$ for constants N_l and N_u derived from Q and constants in $\mathcal{A}$. Here $|S|$ denotes the cardinality of a set S. We refer to Q_l and Q_u as *lower and upper envelopes* of Q under $\mathcal{A}$, respectively. Intuitively, envelopes approximate Q: they guarantee a constant approximation bound, and are boundedly evaluable under $\mathcal{A}$.

Envelopes do not always exist. This motivates us to study *the upper and lower envelopes problems*, denoted by UEP and LEP, respectively. Given a query Q that is not boundedly evaluable under $\mathcal{A}$, UEP (resp. LEP) is to determine whether there exists an upper (resp. lower) envelope of Q under $\mathcal{A}$. To avoid the high complexity of checking BEP, we study envelopes that are covered by $\mathcal{A}$ when Q is in CQ, UCQ or $\exists$FO$^+$. We establish the complexity of UEP and LEP for CQ, UCQ, $\exists$FO$^+$and FO, from NP-complete to undecidable.

Bounded query specialization. The other approach is by specializing queries to achieve bounded evaluability. A query Q in an e-commerce system often comes with a set X of parameters (variables) indicating, *e.g.,* price range and make of a product, which are expected to be instantiated with values of users' choice before Q is executed. Personalized searches of Graph Search [16] are also parameterized queries in which a variable for "person" (*i.e.,* "me") is instantiated by users of the query. We refer to $Q(\bar{x} = \bar{c})$ as a *specialized query* of Q, when a tuple $\bar{x}$ of parameters of X is instantiated with constants $\bar{c}$, referred to as a *valuation* of $\bar{x}$.

We study *the query specialization problem*, denoted by QSP. Given a positive integer k and a query Q that is not boundedly evaluable under $\mathcal{A}$ and comes with a set X of parameters, it is to decide whether there exists a tuple $\bar{x}$ of at most k parameters in X such that $Q(\bar{x} = \bar{c})$ is covered by $\mathcal{A}$ for all valuations $\bar{c}$ of $\bar{x}$, and hence, boundedly evaluable. We provide the complexity of QSP for CQ, UCQ, $\exists$FO$^+$and

FO, ranging over NP-complete, Π_2^p-complete and undecidable. Better still, when $\mathcal{A}$ and a query Q in FO satisfy certain conditions, Q can always be boundedly specialized.

Summary. We study bounded evaluability for computing exact query answers and approximate query answers. We identify several problems for bounded evaluability, and develop their complexity bounds. The complexity results help practitioners assess the difficulty of the bounded evaluability analysis for practical query classes. We also provide characterizations for these problems, to help practitioners develop efficient query plans. Observe that for parameterized queries, it is an *one-time* cost to compute envelopes and bounded specialized queries, although intractable, since these queries remain unchanged and only their parameters are instantiated with different values. The computation can be conducted *offline* when developing the queries.

A variety of (syntactic) characterizations, algorithms and reductions are used to prove these results. Some of the proofs are highly nontrivial. In particular, under access constraints, the satisfiability and containment analyses of queries are a departure from the classical Homomorphism Theorem [13] for CQ and the characterization of [32] for UCQ. These have to be revisited to deal with challenges analogous to what indefinite databases introduce [27,28,34].

Related work. We classify previous work as follows.

Scale independence. The study of bounded evaluability is motivated by the idea of scale independence [6]. The latter aims to guarantee that a bounded amount of work is required to execute all queries in an application, regardless of the size of the underlying data. To enforce scale independence, users may specify bounds on the amount of data accessed and the size of intermediate results; when more data is needed, only top-k tuples are retrieved to meet the bounds [5].

The idea was formalized in [17]. A query Q is called *scale independent* in a dataset D *w.r.t.* a bound M if there is $D_Q \subseteq D$ such that $Q(D) = Q(D_Q)$ and $|D_Q| \leq M$. Access constraints were introduced in [17]. A notion of $\bar{x}$-scale independence was also proposed in [17], to characterize queries $Q(\bar{x}, \bar{y})$ that, for all databases D that satisfy access constraints and for each tuple $\bar{a}$ of values for $\bar{x}$, $Q(\bar{x} = \bar{a}, D)$ can be computed in time dependent on $\mathcal{A}$ and Q only. It showed that x-scale independence is undecidable for FO, and developed syntactic rules as a sufficient condition for deciding the x-scale independence of FO queries under access constraints.

When $\bar{x}$ is empty, *i.e.,* when no instantiation of $\bar{x}$ is required, $\bar{x}$-scale independence was studied in [12], referred to as *effective boundedness*. The notion of effective boundedness is based on *a restricted form* of query plans in which data is fetched before any relational operations. It showed that it is in PTIME to decide whether a CQ Q is effectively bounded under $\mathcal{A}$, *i.e.,* it has a restricted query plan. With real-life data, the approach was experimentally evaluated for CQ in [12] and for graph pattern queries in [11].

This work extends the prior work in the following. (1) We extend access constraints of [12,17], by allowing cardinality bounds to be specified by a (sublinear) function in the size of the underlying data. (2) While [17] has mostly focused on scale independence in *a given database*, we focus on bounded evaluability on *all databases* that satisfy access constraints, like x-scale independence. We also give characterizations for bounded evaluability of queries for various fragments of FO. (3) We study generic query plans that are not allowed by [12]. To see the difference, BEP is $\mathsf{EXPSPACE}$-hard for CQ, as opposed to PTIME for its effective boundedness [12]. To cope with the high complexity of BEP, we give an effective syntax for boundedly evaluable CQ, which is not studied in [12,17]. (4) When exact query answers are beyond reach, we study approximate query answering based on bounded evaluability, which were not considered in [12,17], except a special case of QSP [12]. (5) In the general setting, BEP and QSP have not been studied for CQ, UCQ and $\exists\mathsf{FO}^+$, and none of CQP, UEP and LEP has been considered in [12,17].

Related to access schema is the notion of access patterns, which require that a relation can only be accessed by providing certain combinations of attribute values. Query processing under limited access patterns has been well studied, *e.g.,* [8,15,29,30]. In contrast, access schemas combine indices and cardinality constraints. Our goal is to characterize what queries are boundedly evaluable with access schema, rather than to study the complexity or executable plans for answering queries under access patterns [8,15,29,30].

Approximate query answering. There has been work on approximate query answering, by means of (1) data synopses that given a query Q on a dataset D, compute $Q(D_s)$ in a synopsis D_s of D, such as histograms [24,25], wavelets [21,35] and sampling [3,7]; (2) budgeted search [4,23,36] that terminates the run of an algorithm when reaching a predefined budget (cost or accuracy) and returns intermediate answers. As opposed to the prior work, the study of bounded evaluability aims to (a) fetch $D_Q \subseteq D$ for each query Q based on access constraints, rather than to use a "one-size fits-all" synopsis to answer all queries posed on D, and (b) guarantee accuracy bound for *non-aggregate* queries.

Closer to our work is query-driven approximation [9,14,19,20] that uses a "cheaper" query Q_a instead of Q and computes $Q_a(D)$ as approximate answers to Q in D, *e.g.,* UCQ for recursive datalog [14], tractable queries for CQ [9,20], and (revised) graph simulation for subgraph isomorphism [19]. Following the absolute approximation scheme of [14], we study boundedly evaluable envelopes (UEP and LEP). The problem studied in [14] is to approximate datalog programs with UCQ; it is very different from UEP and LEP considered in this work, which aim to find bounded evaluable envelopes for various FO fragments under access constraints.

Related to specialized queries are query suggestion [26] and parameterized queries, which instantiate parameters with values possibly from a list of suggested keywords. Related to QSP is the $\bar{x}$-controllability problem studied in [17], to find a minimum set $\bar{x}$ of variables in a query Q such that Q can be verified $\bar{x}$-scale independent by the syntactic rules of [17]. It differs from QSP in that $\bar{x}$-scale independence is defined by syntactic rules, as opposed to covered queries. Hence for FO, the $\bar{x}$-controllability problem is in NP, while QSP is undecidable. A special case of QSP was also studied in [12] for CQ, when all variables of Q are treated as parameters. It is based on effective boundedness, as opposed to bounded evaluability. In addition, we also study QSP for UCQ and $\exists\mathsf{FO}^+$, which are not considered in [12].

Organization. Access constraints and bounded evaluability are defined in Section 2. We study the bounded evaluability of queries in Section 3. For approximate query answering, we investigate boundedly evaluable envelopes in Section 4, and bounded query specialization in Section 5. Open problems for future work are identified in Section 6.

2. BOUNDEDLY EVALUABLE QUERIES

We next define access constraints, query plans and boundedly evaluable queries over a relational schema.

A relational schema $\mathcal{R}$ consists of a collection of relation schemas $(R_1, \ldots, R_n)$, where each relation schema R_i has a fixed set of attributes. We assume a countably infinite domain $\mathbf{D}$ of data values, on which instances of $\mathcal{R}$ are defined. For an instance D of $\mathcal{R}$, we use $|D|$ to denote its size, measured as the total *number of tuples* in D.

Query classes. We study the following queries [2].

- Conjunctive queries (CQ), 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 quantification $\exists$.
- Unions of conjunctive queries (UCQ) of the form $Q = Q_1 \cup \cdots \cup Q_k$, where for all $i \in [1, k]$, Q_i is in CQ, referred to as a CQ *sub-query* of Q.
- Positive existential FO queries ($\exists\mathsf{FO}^+$, SPJU of select-project-join-union queries), built from relation atoms and equality atoms by closing under $\wedge$, $\vee$ and $\exists$. For a query Q in $\exists\mathsf{FO}^+$, a CQ *sub-query* of Q is a CQ subquery in the UCQ equivalence of Q.
- First-order logic queries (FO), built from atomic formulas by using $\wedge$, $\vee$, negation $\neg$, $\exists$ and $\forall$.

If $\bar{x}$ is the tuple of free variables of Q, we will 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 $\{\bar{a} \in \mathit{adom}(D)^m \mid D \models Q(\bar{a})\}$, where the active domain, $\mathit{adom}(D)$, consists of all constants appearing in D or Q.

Access schema. An *access schema* $\mathcal{A}$ over a relational schema $\mathcal{R}$ is a set of *access constraints* of the form:

$$R(X \to Y, N),$$

where R is a relation schema in $\mathcal{R}$, X and Y are sets of attributes of R, and N is a natural number.

A relation instance D of R *satisfies* the constraint if

- for any X-value $\bar{a}$ in D, $|D_Y(X = \bar{a})| \leq N$, where $D_Y(X = \bar{a}) = \{t[Y] \mid t \in D,\ t[X] = \bar{a}\}$; and
- there exists an *index on* X *for* Y that given an X-value $\bar{a}$, retrieves $D_Y(X = \bar{a})$.

For instance, ψ_1–ψ_4 given in Example 1.1 together with their indices are access constraints. An access constraint is a combination of a cardinality constraint and an index on X for Y. It tells us that given any X-value, there exist at most N distinct corresponding Y-values, and these Y values can be efficiently retrieved by using the index.

We say that D *satisfies* access schema $\mathcal{A}$, denoted by $D \models \mathcal{A}$, if D satisfies all the constraints in $\mathcal{A}$.

Query plans. To define boundedly evaluable queries, we first present corresponding query plans. Consider a query Q in the relational algebra over schema $\mathcal{R}$, defined in terms of projection operator π, selection σ, Cartesian product $\times$, union $\cup$, set difference $-$ and renaming ρ (see, *e.g.,* [2] for details). A *query plan* for Q is a sequence

$$\xi(Q, \mathcal{R}): \ T_1 = \delta_1, \ \ldots, \ T_n = \delta_n,$$

such that (1) for all instances D of $\mathcal{R}$, $T_n = Q(D)$, and (2) for all $i \in [1, n]$, δ_i is one of the following:

- $\{a\}$, where a is a constant in Q; or
- $\mathsf{fetch}(X \in T_j, R, Y)$, where $j < i$, and T_j has attributes X; for each $\bar{a} \in T_j$, it retrieves $D_{XY}(X = \bar{a})$ from D, and returns $\bigcup_{\bar{a} \in T_j} D_{XY}(X = \bar{a})$; or
- $\pi_Y(T_j)$, $\sigma_C(T_j)$ or $\rho(T_j)$, for $j < i$, a set Y of attributes in T_j, and condition C defined on T_j; or
- $T_j \times T_k$, $T_j \cup T_k$ or $T_j - T_k$, for $j < i$ and $k < i$.

The result $\xi(D)$ of applying $\xi(Q, \mathcal{R})$ to D is T_n.

A query plan $\xi(Q, \mathcal{R})$ is said to be *boundedly evaluable under an access schema* $\mathcal{A}$ if (1) for each $\mathsf{fetch}(X \in T_j, R, Y)$ in it, there exists a constraint $R(X \to Y', N)$ in $\mathcal{A}$ such that $Y \subseteq X \cup Y'$, and (2) the length n of $\xi(Q, \mathcal{R})$ (*i.e.,* the number of operations) is bounded by an exponential in $|\mathcal{R}|$, $|\mathcal{A}|$ and $|Q|$, which are the sizes of $\mathcal{R}$, $\mathcal{A}$ and Q, respectively, *independent of* dataset D. Indeed, a query plan longer than exponential in $|\mathcal{R}|$, $|\mathcal{A}|$ and $|Q|$ is hardly practical.

Intuitively, if $\xi(Q, \mathcal{R})$ is boundedly evaluable under $\mathcal{A}$, then for all instances D of $\mathcal{R}$ that satisfy $\mathcal{A}$, $\xi(Q, \mathcal{R})$ tells us how to fetch $D_Q \subseteq D$ with the indices in $\mathcal{A}$ such that $Q(D) = Q(D_Q)$, where D_Q is the set of all tuples fetched from D by following $\xi(Q, \mathcal{R})$. Better still, D_Q is *bounded*: $|D_Q|$ is determined by Q and constants in $\mathcal{A}$ only. Moreover, the time for identifying and fetching D_Q also depends on Q and $\mathcal{A}$ only (assuming that given an X-value $\bar{a}$, it takes $O(N)$ time to fetch $D_{XY}(X = \bar{a})$ in D with the index in $R(X \to Y, N)$). For instance, a boundedly evaluable query plan for Q_0 is given in Example 1.1 under access constraints ψ_1–ψ_4.

Boundedly evaluable queries. Consider a query Q in a language $\mathcal{L}$ and an access schema $\mathcal{A}$, both over the same relational schema $\mathcal{R}$. We say that Q is *boundedly evaluable under* $\mathcal{A}$ if it has a boundedly evaluable query plan $\xi(Q, \mathcal{R})$ under $\mathcal{A}$ such that in each $T_i = \delta_i$ of $\xi(Q, \mathcal{R})$,

- if $\mathcal{L}$ is CQ, then δ_i is a fetch, π, σ, $\times$ or ρ operation;
- if $\mathcal{L}$ is UCQ, δ_i can be fetch, π, σ, $\times$ or ρ, and there is $k \leq |Q|$ such that the last $k - 1$ operations of $\xi(Q, \mathcal{R})$ are $\cup$, and $\cup$ does not appear anywhere else in $\xi(Q, \mathcal{R})$;
- if $\mathcal{L}$ is $\exists\mathsf{FO}^+$, then δ_i is fetch, π, σ, $\times$, $\cup$ or ρ; and
- if $\mathcal{L}$ is FO, δ_i can be fetch, π, σ, $\times$, $\cup$, $-$ or ρ

One can verify the following: if Q is boundedly evaluable under $\mathcal{A}$, then for *all* instances D of $\mathcal{R}$ that satisfy $\mathcal{A}$, there exists $D_Q \subseteq D$ such that (a) $Q(D_Q) = Q(D)$; (b) the time for identifying and fetching D_Q is determined by Q and $\mathcal{A}$ only; and (c) the size $|D_Q|$ is also *independent of* $|D|$.

General access constraints. We also study access constraints in its general form, defined as follows:

$$R(X \to Y, s(\cdot)),$$

where $s(\cdot)$ is a (sublinear) function in $|D|$.

An instance D of R *satisfies* the constraint if for any given X-value $\bar{a}$, we can retrieve $D_Y(X = \bar{a})$ from D by using an index on X for Y, such that $|D_Y(X = \bar{a})| \leq s(|D|)$.

That is, $|D_Y(X = \bar{a})|$ is bounded by a function in $|D|$, *e.g.,* $\log(|D|)$, rather than by a constant. We refer to these as access constraints *with non-constant cardinality.* Constraints $R(X \to Y, N)$ are a special form when $s(\cdot)$ is a constant N, and are referred to access constraints with *constant cardinality* or simply access constraints. Access constraints with non-constant cardinality are easier to be satisfied, and still allow us to query big data by accessing a small fraction D_Q of the data, although $|D_Q|$ is no longer independent of $|D|$.

To simplify the discussion, we focus on access constraints $R(X \to Y, N)$ with constant cardinality in the sequel. Nonetheless, the characterizations and complexity results of Section 3 remain intact on access constraints with non-constant cardinality, as long as function $s(\cdot)$ is PTIME computable. Similarly, the results on QSP (Section 5) also hold in the presence of the general access constraints.

3. DECIDING BOUNDED EVALUABILITY

We study *the bounded evaluability problem*, denoted by $\mathsf{BEP}(\mathcal{L})$ for a query class $\mathcal{L}$ and stated as follows:

- INPUT: A relational schema $\mathcal{R}$, an access schema $\mathcal{A}$ over $\mathcal{R}$ and a query $Q \in \mathcal{L}$ over $\mathcal{R}$.
- QUESTION: Is Q boundedly evaluable under $\mathcal{A}$?

While BEP(FO) is undecidable [17], we show that for several practical fragments of FO, BEP is decidable. However, the complexity bounds of BEP for these query classes are rather high (Section 3.1). To cope with these, we develop an effective syntax for boundedly evaluable queries in CQ. The syntax is given in terms of a notion of covered queries, which can be checked in PTIME. We extend the notion of covered queries to UCQ and $\exists\mathsf{FO}^+$, to characterize their boundedly evaluable queries. We also provide complexity for deciding whether their queries are covered (Section 3.2).

3.1 Characterizing Bounded Evaluability

No matter how desirable, it is nontrivial to decide whether a query is boundedly evaluable, even for CQ.

Example 3.1: (1) Consider an access schema $\mathcal{A}_1$ and a query Q_1 defined over a relation schema $R_1(A, B, E, F)$:

$$\mathcal{A}_1 = \{\varphi_1 = R_1(A \to B, N_1),\ \varphi_2 = R_1(E \to F, N_2)\},$$
$$Q_1(x, y) = \exists x_1, x_2\big(R(x_1, x, x_2, y) \wedge x_1 = 1 \wedge x_2 = 1\big).$$

Under $\mathcal{A}_1$, Q_1 is seemingly boundedly evaluable: given an instance D_1 of schema R_1, values $x_1 = 1$ and $x_2 = 2$, we can extract x values from D_1 by using φ_1, and y values by φ_2. However, there exists no bounded query plan for Q_1: $\mathcal{A}_1$ does not provide us with indices to check whether these x and y values come from the same tuples in D_1.

(2) Consider $\mathcal{A}_2$ and Q_2 defined on $R_2(A, B)$:

$$\mathcal{A}_2 = \{\varphi_3 = R_2(A \to B, 1)\},$$
$$Q_2(x) = \exists x_1, x_2\big(R_2(x, x_1) \wedge R_2(x, x_2) \wedge x_1 = 1 \wedge x_2 = 2\big).$$

Query Q_2 is boundedly evaluable under $\mathcal{A}_2$, although $\mathcal{A}_2$ does not help us retrieve x values from an instance D_2 of R_2. To see why Q_2 is bounded, note that given any x value, it is impossible to find both $(x, 1)$ and $(x, 2)$ in D_2 that satisfies $\mathcal{A}_2$, because of φ_3. Therefore, $Q_2(D_2) = \emptyset$, *i.e.,* Q_2 is not satisfiable by instances D_2 of R_2 that satisfy $\mathcal{A}_2$. Hence a query plan for empty query suffices to answer Q_2 in D_2.

(3) Consider $\mathcal{A}_3$ and Q_3 defined on $R_3(A, B, C)$:

$$\mathcal{A}_3 = \{\varphi_4 = R_3(\emptyset \to C, 1),\ \varphi_5 = R_3(AB \to C, N)\},$$
$$Q_3(x, y) = \exists x_1, x_2, z_1, z_2, z_3\big(R_3(x_1, x_2, x) \wedge R_3(z_1, z_2, y) \wedge R_3(x, y, z_3) \wedge x_1 = 1 \wedge x_2 = 1\big).$$

At first glance, Q_3 is not boundedly evaluable under $\mathcal{A}_3$, since $\mathcal{A}_3$ does not help us check $R(z_1, z_2, y)$. However, Q_3 is "$\mathcal{A}_3$-equivalent" to Q'_3, *i.e.,* for any instance D_3 of R_3, if $D_3 \models \mathcal{A}_3$, then $Q_3(D_3) = Q'_3(D_3)$, where

$$Q'_3(x, x) = R_3(1, 1, x) \wedge R_3(x, x, x).$$

Query Q'_3 is boundedly evaluable under $\mathcal{A}_3$. Hence, Q_3 is boundedly evaluable under $\mathcal{A}_3$ since a boundedly evaluable query plan for Q'_3 is also a query plan for Q_3.

To see that Q_3 is "$\mathcal{A}_3$-equivalent" to Q'_3, observe the following: for any instance D_3 that satisfies $\mathcal{A}_3$, (a) by φ_4, x, y and z_3 must take the same (unique) value c_0 from D_3, which can be fetched by using the index built for φ_4; hence $R_3(x, y, z_3)$ becomes $R_3(x, x, x)$; and (b) $\exists z_1, z_2(R_3(1, 1, x) \wedge R_3(z_1, z_2, y))$ is equivalent to $R_3(1, 1, x)$; thus $R_3(z_1, z_2, y)$ can be removed. Moreover, Q'_3 is boundedly evaluable under $\mathcal{A}_3$ since by φ_5, we can check whether $(1, 1, x)$ and (x, x, x) are in D_3 when $x = c_0$, using the index for φ_5. □

Impact of access constraints. The complications are introduced partly by access constraints. Consider an access schema $\mathcal{A}$ and a query Q, both defined over the same relational schema $\mathcal{R}$. We say that Q is $\mathcal{A}$-*satisfiable* if there exists an instance D of $\mathcal{R}$ such that $D \models \mathcal{A}$ and $Q(D) \neq \emptyset$.

When Q is a query in CQ, it is in PTIME to decide whether there exists D such that $Q(D) \neq \emptyset$ (satisfiability; cf. [2]). In contrast, $\mathcal{A}$-satisfiability is intractable for CQ.

Lemma 3.2: It is NP-complete to decide whether a query in CQ is $\mathcal{A}$-satisfiable for an access schema $\mathcal{A}$. □

To prove this, we need the following notation. Consider a tableau (T_Q, u) representing a CQ Q (see, *e.g.,* [2]). A valuation θ of (T_Q, u) is a mapping from variables in T_Q to (not necessarily distinct) constants in $\mathbf{D}$. We use $\theta(T_Q)$ to denote the instance obtained by applying θ to variables in T_Q. We call $\theta(T_Q)$ an $\mathcal{A}$-*instance* of Q if $\theta(T_Q) \models \mathcal{A}$. There are possibly exponentially many $\mathcal{A}$-instances of Q up to isomorphism, analogous to representative instances in indefinite data [27, 28, 34]. This is why the $\mathcal{A}$-satisfiability of CQ is more intriguing to check than the satisfiability.

Proof sketch. For the upper bound, we give an NP algorithm that, given (T_Q, u) and $\mathcal{A}$, (a) guesses a valuation θ of tableau (T_Q, u), and (2) checks whether $\theta(T_Q) \models \mathcal{A}$ and $\theta(u)$ is well defined; it returns true if so.

The lower bound is verified by reduction from 3SAT. Given a propositional formula ψ, 3SAT decides whether ψ is satisfiable. It is known to be NP-complete (cf. [31]). □

Recall that query containment and equivalence are NP-complete for CQ, by the Homomorphism Theorem [13]. These classical results on containment and equivalence of CQ no longer hold in the presence of an access schema $\mathcal{A}$. More specifically, we say that a query Q_1 is $\mathcal{A}$-*contained* in query Q_2, denoted by $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$, if for all instances D of $\mathcal{R}$ such that $D \models \mathcal{A}$, $Q_1(D) \subseteq Q_2(D)$. We say that Q_1 and Q_2 are $\mathcal{A}$-*equivalent*, denoted by $Q_1 \equiv_{\mathcal{A}} Q_2$, if $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$ and $Q_2 \sqsubseteq_{\mathcal{A}} Q_1$. Then for CQ, the $\mathcal{A}$-containment and $\mathcal{A}$-equivalence problems are Π_2^p-complete, rather than NP-complete. That is, the presence of access constraints makes the containment and equivalence analyses harder for CQ.

Lemma 3.3: For access schema $\mathcal{A}$ and queries Q_1 and Q_2 in CQ, (1) $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$ if and only if either Q_1 is not $\mathcal{A}$-satisfiable, or for all $\mathcal{A}$-instances $\theta(T_Q)$ of Q_1, $\theta(u) \in Q_2(\theta(T_Q))$; and (2) it is Π_2^p-complete to decide (a) whether $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$ and (b) whether $Q_1 \equiv_{\mathcal{A}} Q_2$. □

Proof sketch. (1) To determine whether $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$, we need to consider (possibly exponentially many) $\mathcal{A}$-instances of Q_1, rather than a "canonical instance" of Q_1 as in [13]. Statement 1 can be verified based on the definition of $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$ and the monotonicity of CQ, since $\mathcal{A}$-instances of Q_1 are instances that satisfy $\mathcal{A}$, on which Q_2 can be applied.

(2) It suffices to show that it is Π_2^p-complete to decide whether $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$, from which the complexity of $Q_1 \equiv_{\mathcal{A}} Q_2$ follows. For the upper bound, we give an Σ_2^p algorithm to determine whether $Q_1 \not\sqsubseteq_{\mathcal{A}} Q_2$, by checking whether Q_1 is $\mathcal{A}$-satisfiable (in NP) and there exists an $\mathcal{A}$-instance $\theta(T_Q)$ of Q_1 such that $\theta(u) \notin Q_2(\theta(T_Q))$ (in Σ_2^p).

The lower bound is verified by reduction from $\forall^*\exists^*$3CNF, which is Π_2^p-complete [33]. The $\forall^*\exists^*$3CNF problem is to decide, given a sentence $\varphi = \forall X \exists Y\, \psi$, whether φ is true, where ψ is an instance of $\mathsf{3SAT}$ defined over $X \cup Y$. The reduction uses $Q_1(X)$ and $Q_2(X)$ to "compute" truth assignments for X such that $\exists Y\, \psi$ is false and true, respectively. □

Complexity. As opposed to $\mathsf{BEP(FO)}$, the BEP analysis is decidable for CQ, although it is highly nontrivial.

Theorem 3.4: $\mathsf{BEP(CQ)}$ is $\mathsf{EXPSPACE}$-complete. □

Proof sketch. The lower bound is verified by reduction from the non-emptiness problem for parameterized regular expressions with certainty semantics, which is shown to be $\mathsf{EXPSPACE}$-complete in [10]. A parameterized regular expression is an extension of conventional regular expressions over alphabet Σ by including variables, which are mapped to symbols in Σ. Given such a parameterized regular expression e, we construct a CQ Q and an access schema $\mathcal{A}$, such that Q has a boundedly evaluable query plan under $\mathcal{A}$ if and only if there exists a string that is in the languages of e under all possible valuations of its variables.

For the upper bound, we develop an $\mathsf{NEXPSPACE}$ algorithm: it guesses a query plan ξ of exponential size, and checks whether ξ is (a) boundedly evaluable under $\mathcal{A}$ and (b) "$\mathcal{A}$-equivalent" to Q, *i.e.,* for all instances D that satisfy $\mathcal{A}$, $\xi(D) = Q(D)$. To check (b), we show that from a boundedly evaluable ξ, an "$\mathcal{A}$-equivalent" CQ Q' can be computed in PTIME in the size of ξ, and checks whether $Q \equiv_{\mathcal{A}} Q'$ by using the algorithm given in the proof of Lemma 3.3. Since $\mathsf{EXPSPACE} = \mathsf{NEXPSPACE}$, $\mathsf{BEP(CQ)}$ is in $\mathsf{EXPSPACE}$. □

Adding unions. We next study BEP for UCQ and $\exists\mathsf{FO}^+$. While $\mathsf{BEP(CQ)}$ is nontrivial, the presence of union makes the bounded evaluability analysis more intriguing. Recall that for two UCQ $Q = \bigcup_{i\in[1,m]} Q_i$ and $Q' = \bigcup_{j\in[1,n]} Q'_j$, $Q \subseteq Q'$ if and only if for each Q_i, there exists Q'_j such that $Q_i \subseteq Q'_j$ [32]. This result of [32] no longer holds when we consider $\mathcal{A}$-containment $\sqsubseteq_{\mathcal{A}}$ under an access schema $\mathcal{A}$.

Example 3.5: Consider a relation schema $R(X)$, an access schema $\mathcal{A}$ with $R(\emptyset \to X, 2)$, and queries below:

$$
\begin{aligned}
&Q(x) = \exists y \big(Q_c(\,) \wedge Q_\psi(x,y)\big),\\
&Q_c(\,) = \exists y_1, y_2 \big(R(y_1) \wedge y_1 = 1 \wedge R(y_2) \wedge y_2 = 0\big),\\
&Q'(x) = Q_1(x) \cup Q_2(x),\\
&Q_1(x) = \exists y (Q_\psi(x,y) \wedge y = 1),\\
&Q_2(x) = \exists y (Q_\psi(x,y) \wedge y = 0),
\end{aligned}
$$

where Q_ψ is a CQ, and Q_c and $\mathcal{A}$ ensure that an R relation encodes Boolean domain $\{0,1\}$. Then one can verify that $Q \sqsubseteq_{\mathcal{A}} Q'$. However, $Q \not\sqsubseteq_{\mathcal{A}} Q_1$ and $Q \not\sqsubseteq_{\mathcal{A}} Q_2$.

As another example, consider $R'(A,B,C)$, $\mathcal{A}'$ consisting of $R'(A \to B, N)$ only, and a query $Q = Q_1 \cup Q_2$, where

$$
\begin{aligned}
&Q_1(y) = \exists x, z (R'(x,y,z) \wedge x = 1),\\
&Q_2(y) = \exists x, z (R'(x,y,z) \wedge x = 1 \wedge z = y).
\end{aligned}
$$

Then under $\mathcal{A}'$, Q_1 and Q are boundedly evaluable, but Q_2 is not. Hence a CQ sub-query of a boundedly evaluable UCQ Q may not be boundedly evaluable itself, as long as it is contained in other sub-queries of Q. □

The lemma below characterizes the bounded evaluability of UCQ under an access schema. It also tells us how to determine whether a query Q in $\exists\mathsf{FO}^+$is boundedly evaluable, since a query in $\exists\mathsf{FO}^+$is equivalent to a query in UCQ.

Lemma 3.6: Under an access schema $\mathcal{A}$, a UCQ Q is boundedly evaluable if and only if Q is $\mathcal{A}$-equivalent to a UCQ $Q' = Q_1 \cup \cdots \cup Q_k$ such that for each $i \in [1,k]$, CQ sub-query Q_i is boundedly evaluable under $\mathcal{A}$. □

We next show that BEP is decidable for UCQ and $\exists\mathsf{FO}^+$.

Corollary 3.7: BEP is $\mathsf{EXPSPACE}$-complete for $\exists\mathsf{FO}^+$. □

Proof sketch. The lower bound follows from Theorem 3.4. For the upper bound, we give an $\mathsf{NEXPSPACE}$ ($\mathsf{EXPSPACE}$) algorithm for checking $\mathsf{BEP}(\exists\mathsf{FO}^+)$, by "decomposing" an $\exists\mathsf{FO}^+$query into a union of "elementary queries" such that their tableaux satisfy $\mathcal{A}$, and by using Lemma 3.6. □

3.2 Effective Syntax

While BEP is decidable for CQ and $\exists\mathsf{FO}^+$, its complexity is too high for us to make practical use of bounded evaluability analysis. This motivates us to develop an effective syntax for their boundedly evaluable queries, with lower complexity.

Effective syntax for CQ. Example 3.1 suggests that to decide whether a CQ Q is boundedly evaluable under an access schema $\mathcal{A}$, we need to check (a) whether Q is "$\mathcal{A}$-equivalent" to a CQ Q' that is boundedly evaluable under $\mathcal{A}$, or (b) whether the indices for constraints in $\mathcal{A}$ "cover" attributes corresponding to variables in Q. We now formalize what queries Q in CQ are "covered by" $\mathcal{A}$, *i.e.,* when the cardinality constraints and indices in $\mathcal{A}$ provide us with sufficient information to fetch tuples for answering Q.

Covered variables. We first look at variables in Q that have to be "covered by" $\mathcal{A}$. Denote by $\mathsf{var}(Q)$ the set of all variables that occur in Q, either free or bound. Assume *w.l.o.g.* that Q is safe, *i.e.,* each variable in $\mathsf{var}(Q)$ is equal to either a variable occurring in a relation atom or a constant in Q. We also assume that queries are satisfiable, *i.e.,* each variable can be equal to at most one constant; and moreover, we assume *w.l.o.g.* that only variables appear in relation atoms of Q, while constants are in equality atoms.

For a variable $x \in \mathsf{var}(Q)$, we denote by $\mathsf{eq}(x,Q)$ the set of all variables in Q that are equal to x as determined by equality atoms of the form $y = z$ in Q, and the transitivity of equality. We define $\mathsf{eq}^+(x,Q)$ as the extension of $\mathsf{eq}(x,Q)$ by including variables y such that $x = y$ can be inferred also from conditions $z = c$ for some constant c (*e.g.,* $x = c$ and $y = c$). We refer to x as a *constant variable* if $\mathsf{eq}(x,Q)$ contains a variable y such that $y = c$ occurs in Q.

A variable x is called *data-dependent* if $\mathsf{eq}(x,Q)$ contains variables that occur in relation atoms of Q, and it is called *data-independent* otherwise. A CQ $Q(\bar{x})$ can be equivalently written as $Q_{dd}(\bar{x}_1) \wedge Q_{di}(\bar{x}_2)$ such that $\bar{x} = (\bar{x}_1, \bar{x}_2)$, $\bar{x}_1$ and $\bar{x}_2$ are disjoint, and Q_{dd} and Q_{di} consist solely of data-dependent and independent variables, respectively.

Example 3.8: Consider a query:

$$Q(x,y,u,v) = R(x,y) \wedge x = 1 \wedge x = y \wedge u = 1 \wedge u = v.$$

Then $\mathsf{eq}(x,Q) = \{x,y\}$ and $\mathsf{eq}^+(x,Q) = \{x,y,u,v\}$. Note that x and y are data-dependent, but u is not, although $u \in \mathsf{eq}^+(x,Q)$. It is to define data-independent variables that we separate $\mathsf{eq}(x,Q)$ from $\mathsf{eq}^+(x,Q)$. □

We next define the set $\mathsf{cov}(Q,\mathcal{A})$ of *variables covered by* $\mathcal{A}$. Intuitively, $\mathsf{cov}(Q,\mathcal{A})$ contains all variables in Q whose values are determined by Q or by $\mathcal{A}$. We define

$$\mathsf{cov}(Q,\mathcal{A}) \;=\; \mathsf{cov}(Q_{dd},\mathcal{A}) \;\cup\; \mathsf{cov}(Q_{di},\mathcal{A}),$$

where $\mathsf{cov}(Q_{di},\mathcal{A}) = \mathsf{var}(Q_{di})$, since the values of such variables do not need to be retrieved from a database D, or to be verified with data in D. We define $\mathsf{cov}(Q_{dd},\mathcal{A})$ inductively, starting from $\mathsf{cov}_0(Q_{dd},\mathcal{A}) = \emptyset$. When $i > 0$, we say that an access constraint $\varphi = R(X \to Y, N)$ is *applicable* to an atom $R(\bar{x},\bar{y},\bar{z})$ in Q_{dd} if the following conditions are satisfied:

- variables $\bar{x}$ correspond to X, and either are already in $\mathsf{cov}_{i-1}(Q,\mathcal{A})$ or are constant variables; and
- $\bar{y}$ corresponds to Y, and there exists a variable y in $\bar{y}$ such that y is not yet in $\mathsf{cov}_{i-1}(Q,\mathcal{A})$.

We define $\mathsf{cov}_i(Q_{dd},\mathcal{A})$ by extending $\mathsf{cov}_{i-1}(Q_{dd},\mathcal{A})$ with the following after each application of a constraint:

- variables in $\mathsf{eq}^+(x,Q_{dd})$ for all constant variables x in $\bar{x}$ that are not already in $\mathsf{cov}_{i-1}(Q,\mathcal{A})$; and
- variables in $\mathsf{eq}^+(y,Q_{dd})$ for each $y \in \bar{y}$.

Note that by using eq^+ instead of eq, we ensure that whenever variable x is covered and $x = c$ holds, then all other variables that are equal to constant c are covered as well. We define $\mathsf{cov}(Q_{dd},\mathcal{A}) = \mathsf{cov}_k(Q_{dd},\mathcal{A})$ when $\mathsf{cov}_k(Q_{dd},\mathcal{A}) = \mathsf{cov}_{k+1}(Q,\mathcal{A})$, *i.e.,* as "the fixpoint".

The lemma below ensures that $\mathsf{cov}(Q,\mathcal{A})$ is well defined, regardless of the order in which constraints in $\mathcal{A}$ are applied.

Lemma 3.9: For any CQ Q and access schema $\mathcal{A}$ over a relational schema $\mathcal{R}$, $\mathsf{cov}(Q,\mathcal{A})$ is uniquely determined and can be computed in PTIME in $|Q|$, $|\mathcal{R}|$ and $|\mathcal{A}|$. □

Covered queries. We are now ready to define covered queries. A CQ $Q(\bar{x})$ is *covered by* $\mathcal{A}$ if

(a) its free variables are covered, *i.e.,* $\bar{x} \subseteq \mathsf{cov}(Q,\mathcal{A})$;

(b) for all non-covered variables $y \notin \mathsf{cov}(Q,\mathcal{A})$, y is non-constant and only occurs once in Q; and

(c) each relation atom $R(\bar{w})$ in Q is *indexed* by $\mathcal{A}$, *i.e.,* there is a constraint $R(Y_1 \to Y_2, N)$ in $\mathcal{A}$ such that (a) all variables in $\bar{w}$ corresponding to attributes Y_1 must be covered, and (b) let $\bar{y}$ be $\bar{w}$ excluding bound variables that only occur once in Q; then each y in $\bar{y}$ corresponds to an attribute in $Y_1 \cup Y_2$.

Intuitively, condition (a) ensures that the values of all free variables of Q are either constants in Q or can be retrieved from a database instance with indices in $\mathcal{A}$. Conditions (b) and (a) together assert that non-covered variables are existentially quantified and do not participate in "joins"; hence, for any instance D of $\mathcal{R}$, $Q(D)$ does not depend on what values these variables take. Condition (c) requires that when we need $t[Y]$ values of an R tuple t to answer Q, the values of all attributes in Y come from the same tuple t and can be retrieved (checked) by using an index in $\mathcal{A}$.

Example 3.10: Query Q_3 of Example 3.1 is covered by $\mathcal{A}_3$: (a) $\mathsf{cov}(Q_3,\mathcal{A}_3) = \{x,y,z_3,x_1,x_2\}$, including all free variables x and y; (b) while z_1 and z_2 are uncovered, they satisfy condition (b), and thus their values has no impact on answers to Q_3; and (c) relations $R(x_1,x_2,x)$ and $R(x,y,z_3)$ are indexed by φ_5, and $R(z_1,z_2,y)$ is indexed by φ_4.

In contrast, query Q_1 of Example 3.1 is not covered by $\mathcal{A}_1$: Q_1 does not satisfy condition (c), since relation atom $R(x_1,x,x_2,y)$ is not indexed by any constraint in $\mathcal{A}_1$.

As another example, query Q_0 of Example 1.1 is covered by $\mathcal{A}_0$ consisting of ψ_1–ψ_4. Indeed, its free variable x_a is covered, non-covered variables cid and class occur only once in Q_0, and all its relation atoms are indexed: Accident by ψ_3, Casualty by ψ_2 and Vehicle by ψ_4. □

Effective syntax. Covered CQ queries provide us with an effective syntax for boundedly evaluable CQ queries. In our experiments with real-life data [12], we find that most boundedly evaluable CQ queries are covered.

Theorem 3.11: For an access schema $\mathcal{A}$ and a CQ Q.

(1) Q is boundedly evaluable under $\mathcal{A}$ if and only if Q is $\mathcal{A}$-equivalent to a CQ Q' that is covered by $\mathcal{A}$;

(2) if Q is covered by $\mathcal{A}$, then Q is boundedly evaluable under $\mathcal{A}$; and

(3) checking whether Q is covered by $\mathcal{A}$ is in PTIME in $|Q|$, $|\mathcal{A}|$ and $|\mathcal{R}|$, where $\mathcal{R}$ is the relational schema over which Q and $\mathcal{A}$ are defined. □

Proof sketch. The proof is a little involved, and needs the following lemmas, which are verified with constructive proofs, *i.e.,* by developing algorithms needed. Consider query plans, an access schema $\mathcal{A}$ and queries over a relational schema $\mathcal{R}$.

(a) Every boundedly evaluable query plan ξ under $\mathcal{A}$ for a CQ determines a CQ Q_ξ such that Q_ξ is covered by $\mathcal{A}$ and for all instances D of $\mathcal{R}$, if $D \models \mathcal{A}$, then when ξ is applied to D, $\xi(D) = Q_\xi(D)$. This is verified by induction on the length of ξ, constructing Q_ξ step by step.

(b) If a CQ Q is covered by $\mathcal{A}$, then Q is boundedly evaluable under $\mathcal{A}$. This is verified by generating a boundedly evaluable query plan ξ for Q, mimicking each step of the evaluation of Q with an operation in ξ.

From Lemmas (a) and (b), statement (1) follows. Statement (2) follows from Lemma (b). Statement (3) follows from Lemma 3.9 and the fact that checking conditions (b) and (c) of covered queries can be done in PTIME. □

Example 3.12: The notion of coverage characterizes what makes a CQ boundedly evaluable. For instance, Q_0 of Example 1.1 is covered by $\mathcal{A}_0$, and Q_3 of Example 3.1 is covered by $\mathcal{A}_3$. As shown earlier, both queries are boundedly evaluable. The characterization is, however, not purely syntactic. Some boundedly evaluable CQ queries may not be covered, but are $\mathcal{A}$-equivalent to a covered query in CQ. For example, Q_2 of Example 3.1 is not covered by $\mathcal{A}_2$: its free variable x is not in $\mathsf{cov}(Q_2,\mathcal{A}_2)$. Nonetheless, Q_2 is $\mathcal{A}_2$-equivalent to a query $Q'_2(x) = (x = 1 \wedge x = 2)$, which is covered by $\mathcal{A}_2$ since its variable is data-independent. □

Effective syntax for $\exists\mathsf{FO}^+$. We now extend the notion of covered queries to $\exists\mathsf{FO}^+$(and hence UCQ). A query Q in $\exists\mathsf{FO}^+$is *covered* by an access schema $\mathcal{A}$ if for each Q_i of its CQ sub-queries, either (a) Q_i is covered, or (b) for all $\mathcal{A}$-instances $\theta(T_Q)$ of Q_i, there is $j \in [1,k]$ such that $\theta(u) \in Q_j(\theta(T_Q))$ and Q_j is covered by $\mathcal{A}$.

Covered queries are also an effective syntax for boundedly evaluable queries in $\exists\mathsf{FO}^+$. Indeed, the corollary below follows from Theorem 3.11 and Lemma 3.6.

Corollary 3.13: (1) An $\exists\mathsf{FO}^+$query is boundedly evaluable under an access schema $\mathcal{A}$ if and only if it is $\mathcal{A}$-equivalent to an $\exists\mathsf{FO}^+$query that is covered by $\mathcal{A}$. (2) Each $\exists\mathsf{FO}^+$query covered by $\mathcal{A}$ is boundedly evaluable under $\mathcal{A}$. □

Deciding coverage. We study the *query coverage problem*, denoted by $\mathsf{CQP}(\mathcal{L})$ and stated as follows.

- INPUT: $\mathcal{R}$, $\mathcal{A}$ and Q as in BEP.
- QUESTION: Is Q covered by $\mathcal{A}$?

In practice, the analysis of CQP helps us syntactically check whether Q is boundedly evaluable under an access schema.

By Theorem 3.11, CQP is in PTIME for CQ, as opposed to $\mathsf{EXPSPACE}$-complete for BEP. It provides us with a tractable syntactic method to check the bounded evaluability of CQ. However, CQP is nontrivial when it comes to UCQ and $\exists\mathsf{FO}^+$, although it is easier than its BEP counterparts.

Theorem 3.14: CQP is

- in PTIME for CQ; and
- Π_2^p-complete for UCQ and $\exists\mathsf{FO}^+$. □

Alternatively, one can define a query Q in $\exists\mathsf{FO}^+$to be covered if each of its CQ sub-query is covered. If so, $\mathsf{CQP}(\mathsf{UCQ})$ is in PTIME and $\mathsf{CQP}(\exists\mathsf{FO}^+)$ is coNP-complete, down from Π_2^p-complete. We opt to adopt a more general notion of covered queries for $\exists\mathsf{FO}^+$, to include most boundedly evaluable UCQ and $\exists\mathsf{FO}^+$queries found in practice.

Proof sketch. We show that CQP is in Π_2^p for $\exists\mathsf{FO}^+$and Π_2^p-hard for UCQ. For the upper bound, we develop an Σ_2^p algorithm that checks whether a query Q in $\exists\mathsf{FO}^+$ is not covered by an access schema. The lower bound is verified by reduction from the $\forall^*\exists^*$3CNF problem; it is a revision of its counterpart given in the proof of Lemma 3.3. □

Generalization. Access constraints with non-constant cardinality (Section 2) do not make our lives harder.

Corollary 3.15: All the results of this section (Theorems 3.11, 3.4 and 3.14, Lemmas 3.2, 3.3, 3.9, 3.6, as well as Corollaries 3.7 and 3.13) also hold under access constraints of the general form $R(X \to Y, s(\cdot))$. □

4. QUERY DRIVEN APPROXIMATION

When a query Q is boundedly evaluable under an access schema $\mathcal{A}$, in all datasets D that satisfy $\mathcal{A}$, we can compute $Q(D)$ by accessing a bounded amount of data. If Q is not boundedly evaluable, however, it may be cost-prohibitive to compute exact answers to Q in D. In light of this, we study how to compute approximate query answers to Q following the absolute approximation scheme of [14]. Below we first present envelopes based on bounded evaluability in Section 4.1. We then study the existence of upper and lower envelopes in Sections 4.2 and 4.3, respectively.

4.1 Boundedly Evaluable Envelopes

Consider an access schema $\mathcal{A}$ and a query Q, both defined over a relational schema $\mathcal{R}$, where Q is in query language $\mathcal{L}$, and Q is not boundedly evaluable under $\mathcal{A}$.

We want to find queries Q_l and Q_u in $\mathcal{L}$ such that

(a) Q_l and Q_u are boundedly evaluable under $\mathcal{A}$; and

(b) for all instances D of $\mathcal{R}$ that satisfy $\mathcal{A}$,
  - $Q_l(D) \subseteq Q(D) \subseteq Q_u(D)$, and
  - $|Q(D) - Q_l(D)| \leq N_l$, $|Q_u(D) - Q(D)| \leq N_u$,

where N_l and N_u are constants derived from Q and constants in $\mathcal{A}$. We refer to Q_u and Q_l as *upper* and *lower envelopes* of Q under $\mathcal{A}$, respectively, and call N_u (resp. N_l) an *approximation bound* of Q_u (resp. Q_l) *w.r.t.* Q.

Intuitively, upper and lower envelopes approximate query Q. Given any instance D of $\mathcal{R}$, as long as $D \models \mathcal{A}$, $Q_u(D)$ and $Q_l(D)$ can be efficiently computed by accessing a bounded amount of data. Better still, $Q_u(D)$ and $Q_l(D)$ are not too far from the exact answers $Q(D)$: $Q_u(D)$ includes all tuples in $Q(D)$, and it has at most N_u tuples that are not in $Q(D)$; moreover, all tuples in $Q_l(D)$ are also in $Q(D)$, and at most N_l tuples in $Q(D)$ are not in $Q_l(D)$.

Example 4.1: Consider a relation schema $R(A, B)$, an access schema $\mathcal{A}$ consisting of a single constraint $R(A \to B, N)$ for a constant N, and two queries in CQ:

$$Q_1(x) = \exists y, z, w\big(R(w, x) \wedge R(y, w) \wedge R(x, z) \wedge w = 1\big);$$
$$Q_2(x, y) = \exists w\big(R(w, x) \wedge R(y, w) \wedge w = 1\big).$$

Then Q_1 is not boundedly evaluable under $\mathcal{A}$. However, it has upper envelope Q_u and lower envelope Q_l:

$$Q_u(x) = \exists y, z\big(R(1, x) \wedge R(x, z)\big),$$
$$Q_l(x) = \exists y, z\big(R(1, x) \wedge R(y, 1) \wedge R(x, y) \wedge R(x, z)\big).$$

Indeed, Q_u and Q_l are covered by $\mathcal{A}$ and are boundedly evaluable. Moreover, for any instance D of R, if $D \models \mathcal{A}$, then $|Q_u(D) - Q_1(D)| \leq N$ and $|Q_1(D) - Q_l(D)| \leq N$.

In contrast, Q_2 is not boundedly evaluable under $\mathcal{A}$, and it has neither upper nor lower envelope. □

As we have seen in Example 4.1, a query may not have upper or lower envelopes, *e.g.*, Q_2. This suggests that we study problems for deciding whether a query Q has envelopes under an access schema, to help us determine whether it is possible to approximate Q with boundedly evaluable queries that warrant constant approximation bounds.

However, the problems for deciding the existence of envelopes for a given query Q are even harder than BEP, the problem for deciding the bounded evaluability of Q. In light of this we consider envelopes of certain syntactic forms, to get lower complexity for the decision problems.

4.2 Deciding Upper Envelopes

We first define upper envelopes of a certain syntactic form, and then study the associated decision problem.

Query relaxation. Assume a relational schema $\mathcal{R}$ over which our queries and access schemas are defined.

A *relaxation* of a CQ $Q(\bar{x}) = \exists\bar{y}\psi(\bar{x}, \bar{y})$ is a CQ $Q'(\bar{x}) = \exists\bar{y}'\psi'(\bar{x}, \bar{y}')$ such that $\bar{y}' \subseteq \bar{y}$, and moreover, every atomic formula in ψ' is an atomic formula in ψ.

For instance, query Q_u given in Example 4.1 is a relaxation of Q_1. Intuitively, Q' is obtained by removing tuples from the tableau representing Q. Note that Q and Q' have the same set of free variables and $Q \subseteq Q'$. Hence $Q \sqsubseteq_{\mathcal{A}} Q'$ for any access schema $\mathcal{A}$ defined over $\mathcal{R}$.

We extend the notion of relaxation to $\exists\mathsf{FO}^+$. A *relaxation* of an $\exists\mathsf{FO}^+$query Q is a query Q' in $\exists\mathsf{FO}^+$such that each CQ sub-query Q'_i of Q' is a relaxation of a CQ sub-query of Q.

Decision problem. The *upper envelope problem* for a query class $\mathcal{L}$, denoted by $\mathsf{UEP}(\mathcal{L})$, is stated as follows.

- INPUT: A relational schema $\mathcal{R}$, an access schema $\mathcal{A}$ over $\mathcal{R}$, and a query $Q \in \mathcal{L}$ over $\mathcal{R}$ that is not boundedly evaluable under $\mathcal{A}$.
- QUESTION: Does there exist an upper envelope Q_u of Q under $\mathcal{A}$? In particular, when $\mathcal{L}$ is CQ, UCQ or $\exists\mathsf{FO}^+$, it is to decide whether there exists Q_u that is a relaxation of Q and is covered by $\mathcal{A}$.

That is, whenever possible, we search for upper envelopes that can be syntactically checked, to reduce the cost of checking their bounded evaluability. By Corollary 3.13, a covered query is boundedly evaluable.

Characterization. What queries can have an upper envelope? We start with a condition that is necessary for the existence of both upper and lower envelopes.

A query Q is *bounded under* $\mathcal{A}$ if there exists a constant c determined by Q and $\mathcal{A}$ such that for *all* instances D of $\mathcal{R}$, if $D \models \mathcal{A}$, then there exists $D_Q \subseteq D$, where

(a) $Q(D_Q) = Q(D)$; and

(b) $|D_Q| \leq c$, *i.e.,* $|D_Q|$ is independent of $|D|$.

Hence, there exists a constant c_r such that $|Q(D)| \leq c_r$.

The notion of boundedness is weaker than the notion of boundedly evaluability. A boundedly evaluable query is also bounded, but a bounded query may *not* be boundedly evaluable, *i.e.,* it does not necessarily have an boundedly evaluable query plan. For instance, query Q_1 of Example 4.1 is bounded, but it is not boundedly evaluable.

Recall that query Q_2 of Example 4.1 is not bounded, and it does not have an envelope. This is not a coincidence. Indeed, boundedness is a necessary condition for a query to have an envelope, as shown by the lemma below.

Lemma 4.2: Under an access schema $\mathcal{A}$,

(a) if a query Q has an (upper or lower) envelope, then Q must be bounded;

(b) a CQ $Q(\bar{x})$ is bounded if and only if all free variables $\bar{x}$ of Q are covered by $\mathcal{A}$; and

(c) a query Q in $\exists\mathsf{FO}^+$is bounded if and only if every CQ sub-query of Q is bounded. □

Proof sketch. If Q has an envelope Q', then Q' is boundedly evaluable and hence for all instances D that satisfy $\mathcal{A}$, $|Q(D)| \leq c$ for a constant c. Thus if Q is not bounded, Q' does not have a constant approximation bound for Q' *w.r.t.* Q. From this statement (a) follows.

Statements (b) and (c) are verified based on the monotonicity of CQ and $\exists\mathsf{FO}^+$. Note that statement (c) only holds for bounded queries. In contrast, for a boundedly evaluable query in $\exists\mathsf{FO}^+$, some of its CQ sub-queries may *not* be boundedly evaluable, as Example 3.5 demonstrates. □

For a CQ Q that is not boundedly evaluable under $\mathcal{A}$, UEP asks whether we can make Q covered by removing relation atoms, and hence removing variables that are not covered by $\mathcal{A}$. For instance, query Q_1 of Example 4.1 has a relation atom $R(y, w)$ with variable y that is not covered. We remove $R(y, w)$ and get an upper envelope Q_u that is covered.

When Q is in $\exists\mathsf{FO}^+$, the lemma below characterizes UEP for $\exists\mathsf{FO}^+$, which can be verified based on the definitions of query relaxations and covered queries for $\exists\mathsf{FO}^+$.

Lemma 4.3: Under an access schema $\mathcal{A}$, a query Q in $\exists\mathsf{FO}^+$has an upper envelope that is a relaxation and covered if and only if for each CQ sub-query Q_i of Q, either Q_i has a covered relaxation, or for any $\mathcal{A}$-instance $\theta(T_Q)$ of Q_i, there exists a covered relaxation Q'_j of a CQ sub-query Q_j such that $\theta(u) \in Q'_j(\theta(T_Q))$. □

Complexity. We next give the complexity of $\mathsf{UEP}(\mathcal{L})$. To make the picture complete, we also study $\mathsf{UEP}(\mathsf{FO})$ in which an upper envelope Q_u is simply defined to be a boundedly evaluable FO query such that $Q \sqsubseteq_{\mathcal{A}} Q_u$ and Q_u has a constant approximation bound *w.r.t.* Q.

While UEP is intractable for CQ and $\exists\mathsf{FO}^+$, its analyses are much simpler than their BEP counterparts.

Theorem 4.4: Under an access schema, UEP is

- NP-complete for CQ;
- Π_2^p-complete for UCQ and $\exists\mathsf{FO}^+$; and
- undecidable for FO. □

Proof sketch. (1) *Lower bounds.* We show that UEP is NP-hard, Π_2^p-hard and undecidable for CQ, UCQ and FO by reductions from $\mathsf{X3C}$, $\forall^*\exists^*$3CNF and the complement of the satisfiability problem for FO, respectively. The $\mathsf{X3C}$ problem (exact cover by 3-sets) is to determine, given a set X with $3q$ elements and a collection C of 3-element subsets of X, whether C contains an exact cover C' of X, *i.e.,* $C' \subseteq C$ such that every element of X occurs in exactly one subset of C'. It is NP-complete (cf. [31]). The satisfiability problem for FO is to decide, given an FO query Q over a relational schema $\mathcal{R}$, whether there is an instance D of $\mathcal{R}$ such that $Q(D) \neq \emptyset$. It is undecidable (cf. [2]).

It should be remarked that while $\mathsf{UEP}(\mathsf{UCQ})$ has the same complexity as $\mathsf{CQP}(\mathsf{UCQ})$, the reduction for UEP is more involved than its counterpart for CQP.

(2) *Upper bounds.* We develop an NP algorithm for checking whether a CQ has a relaxation that is covered by $\mathcal{A}$, based on Theorem 3.11. Capitalizing on Lemma 4.3, we develop an Σ_2^p algorithm to check whether a query in $\exists\mathsf{FO}^+$does *not* have a relaxation that is covered by $\mathcal{A}$. □

4.3 Deciding Lower Envelopes

Analogous to the analysis of upper envelopes, we study lower envelopes of a certain syntactic form.

Query expansion. Assume a positive integer k. A *k-expansion* of a CQ $Q(\bar{x}) = \exists\bar{y}\psi(\bar{x}, \bar{y})$ is a CQ $Q'(\bar{x}) = \exists\bar{y}'\psi'(\bar{x}, \bar{y}')$ such that $\bar{y} \subseteq \bar{y}'$, every atomic formula in ψ is an atomic formula in ψ', and moreover, ψ' contains at most k relation atoms that do not occur in ψ.

Intuitively, let (T_Q, u) be the tableau representation of Q, and T'_Q be a tableau obtained by adding at most k additional tuples to T_Q. Then Q' is a CQ represented by (T'_Q, u). For instance, query Q_l given in Example 4.1 is an 1-expansion of query Q_1. Observe that $Q' \subseteq Q$ and $Q' \sqsubseteq_{\mathcal{A}} Q$ for any access schema $\mathcal{A}$ that is defined over the same relational schema $\mathcal{R}$ on which queries Q and Q' are defined.

We define a *k-expansion* of a query Q in $\exists\mathsf{FO}^+$ to be a query Q' in $\exists\mathsf{FO}^+$such that each CQ sub-query of Q' is a k-expansion of a CQ sub-query of Q.

Decision problem. We now state the *lower envelope problem* for a query class $\mathcal{L}$, denoted by $\mathsf{LEP}(\mathcal{L})$.

- INPUT: $\mathcal{R}$, $\mathcal{A}$, Q as in UEP, and a natural number k.
- QUESTION: Does there exist a lower envelope Q_l of Q under $\mathcal{A}$ that is $\mathcal{A}$-satisfiable? In particular, when $\mathcal{L}$ is CQ, UCQ or $\exists$FO$^+$, it is to decide whether there exists a lower envelope Q_l that is a k-expansion of Q and is covered by $\mathcal{A}$.

We refer to Q_l as a *k-expansion lower envelope.*

We require Q_l to be $\mathcal{A}$-satisfiable to rule out "trivial" lower envelopes. Note that when a CQ Q is bounded under $\mathcal{A}$, empty query $Q_\emptyset$ would have been a lower envelope of Q. Such a trivial envelope is not very useful. We do not impose the condition on upper envelopes, since an upper envelope Q_u is guaranteed $\mathcal{A}$-satisfiable. Indeed, UEP is studied for Q that is not boundedly evaluable under $\mathcal{A}$; hence Q must be $\mathcal{A}$-satisfiable. By $Q \sqsubseteq_\mathcal{A} Q_u$, Q_u is also $\mathcal{A}$-satisfiable.

Characterization. For a CQ Q that is not boundedly evaluable, LEP is to decide whether we can make Q covered by adding additional relation atoms. Intuitively, when Q contains variables that are not covered, we add relation atoms to make them covered, as illustrated by Q_l of Example 3.10. When Q contains relation atoms $R(\bar{y})$ that are not indexed by $\mathcal{A}$ (see the definition of covered queries in Section 3.1), sometimes we can "split" $R(\bar{y})$ into $R(\bar{y}_1) \wedge \ldots \wedge R(\bar{y}_n)$ such that $\bar{y} = (\bar{y}_1, \ldots, \bar{y}_n)$ and each $R(\bar{y}_i)$ is indexed.

Example 4.5: Consider a relation schema $R(A, B, C)$, an access schema $\mathcal{A}$ and a CQ Q defined as follows:

$$\mathcal{A} = \{R(A \to B, N), R(B \to C, 1)\},$$
$$Q(x, y) = R(1, x, y).$$

Then Q is not covered by $\mathcal{A}$, since $R(1, x, y)$ is not indexed by $\mathcal{A}$. Nonetheless, its 1-expansion below is covered:

$$Q'(x, y) = \exists z_1, z_2 \big(R(1, x, z_1) \wedge R(z_2, x, y)\big).$$

One can verify that Q' is indexed and $Q' \equiv_\mathcal{A} Q$. □

For query Q in $\exists$FO$^+$, a characterization for the existence of lower envelopes is given as follows, which can be verified by using the definitions of covered queries and k-expansions.

Lemma 4.6: Under an access schema $\mathcal{A}$, a query Q in $\exists$FO$^+$has a k-expansion lower envelope if and only if (a) Q is bounded under $\mathcal{A}$, and (b) there exists a CQ sub-query Q_i of Q such that it has a k-expansion that is covered by $\mathcal{A}$ and is $\mathcal{A}$-satisfiable. □

Complexity. Compared to UEP($\mathcal{L}$), LEP($\mathcal{L}$) has a lower complexity when $\mathcal{L}$ is UCQ or $\exists$FO$^+$.

Theorem 4.7: Under an access schema $\mathcal{A}$, LEP is

- NP-complete for CQ and UCQ;
- DP-complete for $\exists$FO$^+$; and
- undecidable for FO. □

Proof sketch. (1) *Lower bounds.* We show that LEP is NP-hard, DP-hard and undecidable for CQ, $\exists$FO$^+$and FO, by reduction from X3C, SAT-UNSAT and the complement of the satisfiability problem for FO, respectively. SAT-UNSAT is to decide, given a pair (φ_1, φ_2) of 3SAT instances, whether φ_1 is satisfiable and φ_2 is not satisfiable. It is DP-complete (cf. [31]). The reduction from SAT-UNSAT makes use of nested union in $\exists$FO$^+$query, which is not supported by UCQ.

(2) *Upper bounds.* Based on Lemmas 4.2 and 4.6, we develop an algorithm to check whether a query has a lower envelope that is a k-expansion, $\mathcal{A}$-satisfiable and covered. It is in NP for UCQ. In contrast, it is in DP for $\exists$FO$^+$since it uses a coNP oracle to check whether Q is bounded, and an NP oracle to check whether Q has a covered k-expansion. □

General constraints. When access constraints with non-constant cardinality are considered, the notion of bounded queries needs to be revised to accommodate cardinality functions, and the results of this section do not carry over directly to access constraints of the general form.

5. BOUNDED QUERY SPECIALIZATION

For a query Q that is not boundedly evaluable, the chances are that Q will become boundedly evaluable when its users instantiate some parameters of Q. This suggests another strategy to process costly queries based on bounded evaluability. As remarked in Section 1, parameterized queries are common in e-commerce systems and personalized searches, and such queries are typically specialized by instantiating some of its parameters when being issued by its users. Below we study QSP, the query specialization problem.

5.1 Query Specialization

We first present (bounded) query specialization.

Specialized queries. First consider $Q(\bar{y}) = \exists \bar{z}\ \psi(\bar{y}, \bar{z})$ in CQ, where ψ is quantifier free, and $\bar{z}$ consists of bound variables. The *parameters* of Q, denoted by X, may include both free variables of $\bar{y}$ and bound variables of $\bar{z}$. Such parameters are typically *designated* by the provider of Q.

A *specialized query* $Q(\bar{x} = \bar{c})$ of Q is defined as $\exists \bar{z}(\psi(\bar{y}, \bar{z}) \wedge \bar{x} = \bar{c})$, where $\bar{x}$ is a tuple of parameters in X, and $\bar{c}$ is a tuple of constants with $|\bar{x}| = |\bar{c}|$. Here we use $|\bar{x}|$ to denote the arity of $\bar{x}$, and refer to $\bar{c}$ as a *valuation* of $\bar{x}$. That is, we specialize Q by instantiating parameters $\bar{x}$.

Example 5.1: Consider query Q defined on relations Accident, Casualty and Vehicle given in Example 1.1:

$$\begin{aligned} Q(x_a) = \exists\ & \mathsf{aid}, \mathsf{date}, \mathsf{district}, \mathsf{cid}, \mathsf{class}, \mathsf{vid}, \mathsf{dri} \\ & \big(\mathsf{Accident}(\mathsf{aid}, \mathsf{district}, \mathsf{date}) \wedge \\ & \mathsf{Casualty}(\mathsf{cid}, \mathsf{aid}, \mathsf{class}, \mathsf{vid}) \wedge \mathsf{Vehicle}(\mathsf{vid}, \mathsf{dri}, x_a)\big). \end{aligned}$$

It has two parameters date and district in X, identified by the designer of Q. Given a valuation (c_1, c_2) of (date, district), the specialized query $Q(\mathsf{date} = c_1, \mathsf{district} = c_2)$ of Q is to find the ages of drivers who were involved in an accident in district c_2 on day c_1. For instance, $Q(\mathsf{date} =$ "1/5/2005", $\mathsf{district} =$ "Queen's Park") is query Q_0 given in Example 1.1.

Under access constraints ψ_1–ψ_4 of Example 1.1, (1) Q is *not* boundedly evaluable itself, since free variable x_a is not covered; but (2) $Q(\mathsf{date} = c_1)$ is boundedly evaluable for *all* valuations c_1 of date; *i.e.,* instantiating a single parameter makes the specialized queries boundedly evaluable. □

For an FO query Q, consider its DNF form: $Q(\bar{y}) = P_1 z_1 \ldots P_n z_n \psi(\bar{y}, \bar{z})$, where P_i is either $\exists$ or $\forall$, and $\bar{z}$ denotes $(z_1, \ldots, z_n)$. Its *parameters* in X may be variables from $\bar{y}$ and $\bar{z}$. A *specialized query* $Q(\bar{x} = \bar{c})$ of Q is defined as $P_1 z_1 \ldots P_n z_n(\psi(\bar{y}, \bar{z}) \wedge \bar{x} = \bar{c})$, where $\bar{x}$ is a tuple of parameters in X, and $\bar{c}$ is a valuation of $\bar{x}$.

Bounded query specialization. Consider query Q that is not boundedly evaluable under an access schema $\mathcal{A}$, with a parameter set X. We say that Q can be *boundedly specialized* *under $\mathcal{A}$ with $\bar{x}$* if $\bar{x}$ is a tuple of parameters from X such that (a) $Q(\bar{x} = \bar{c})$ is boundedly evaluable under $\mathcal{A}$ for *all valuations* $\bar{c}$ of $\bar{x}$, and (b) there exists at least one valuation $\bar{c}$ of $\bar{x}$ such that $Q(\bar{x} = \bar{c})$ is $\mathcal{A}$-satisfiable.

Intuitively, condition (a) asks for $Q(\bar{x} = \bar{c})$ to be generic regardless of what valuations are used, and condition (b) requires the specialized query to be sensible.

Some queries Q may not be boundedly specialized. For instance, recall query Q from Example 5.1. If its set X of parameters consists of district only, one can verify that Q may not be boundedly specialized under constraints ψ_1–ψ_4. Moreover, if Q can be boundedly instantiated, we naturally want to instantiate a minimum set of parameters in X.

Decision problem. Hence we study the *query specialization problem*, denoted by $\mathsf{QSP}(\mathcal{L})$ for a query language $\mathcal{L}$.

- INPUT: A relational schema $\mathcal{R}$, an access schema $\mathcal{A}$ over $\mathcal{R}$, a query $Q \in \mathcal{L}$ defined over $\mathcal{R}$ that is not boundedly evaluable under $\mathcal{A}$, a set X of parameters in Q, and a natural number k.
- QUESTION: Can Q be boundedly specialized under $\mathcal{A}$ with a tuple $\bar{x}$ from X such that $|\bar{x}| \leq k$? In particular, when $\mathcal{L}$ is CQ, UCQ or $\exists\mathsf{FO}^+$, it is to decide whether there exists $\bar{x}$ such that $|\bar{x}| \leq k$ and $Q(\bar{x} = \bar{c})$ is covered by $\mathcal{A}$ for all valuations $\bar{c}$ of $\bar{x}$.

The study of QSP aims to help us decide what access schema to maintain and what parameters to instantiate, to make specialized queries boundedly evaluable.

When $\mathcal{L}$ is CQ, UCQ or $\exists\mathsf{FO}^+$, we ask for specialized queries $Q(\bar{x} = \bar{c})$ that are covered by $\mathcal{A}$, to reduce the cost of the QSP analysis. By Corollary 3.13, $Q(\bar{x} = \bar{c})$ is boundedly evaluable under $\mathcal{A}$. Without the syntactic restriction, $\mathsf{QSP}(\mathcal{L})$ has complexity higher than $\mathsf{BEP}(\mathcal{L})$ when $\mathcal{L}$ is, *e.g.,* CQ, and is too costly to be practical.

Remark. Both QSP and LEP aim to restrict a query Q and make it boundedly evaluable. However, QSP approaches bounded evaluability by instantiating parameters, while LEP is by imposing additional relation atoms on Q. Moreover, LEP requires that $|Q(D) - Q_l(D)| \leq N_l$ with a constant N_l for all instances D that satisfy $\mathcal{A}$. In light of this, Q has to be bounded to get a lower envelope, whereas this is not required by QSP. As will be seen shortly, $\mathsf{QSP}(\mathcal{L})$ and $\mathsf{LEP}(\mathcal{L})$ have different complexity for UCQ and $\exists\mathsf{FO}^+$.

5.2 Deciding Bounded Specialization

We next study the complexity of $\mathsf{QSP}(\mathcal{L})$. It is nontrivial to identify parameters $\bar{x}$ of Q for instantiation and make specialized $Q(\bar{x} = \bar{c})$ boundedly evaluable.

Example 5.2: Consider a relational schema $\mathcal{R}$, an access schema $\mathcal{A}$ and a CQ Q over $\mathcal{R}$: (1) $\mathcal{R}$ consists of $R_i(A, B_1, B_2, B_3)$ for $i \in [1, n]$, (2) $\mathcal{A}$ defines 4 constraints on each R_i: $R_i(A \to (B_1, B_2, B_3), 1)$, $R_i(B_1 \to A, 1)$, $R_i(B_2 \to A, 1)$ and $R_i(B_3 \to A, 1)$; and (3) Q is

$$\exists \bar{y}, \bar{z}\big(\bigwedge_{i\in[1,n]} R_i(1,1,1,1) \land \bigwedge_{i\in[1,n]} R_i(y_i, z_{i1}, z_{i2}, z_{i3})\big).$$

One can verify that the Boolean query $Q()$ is not boundedly evaluable under $\mathcal{A}$. Now let X be $\bar{y}$ and k be a positive integer. We want to know whether Q can be boundedly specialized with $\bar{x}$ from X and $|\bar{x}| \leq k$.

In the proof of Theorem 5.3, we use $\mathcal{R}$, $\mathcal{A}$ and Q to encode an instance of the minimum set cover problem (MSC). Given a collection C of subsets of a finite set S and a natural number k, MSC is to decide whether there exists a cover C' of C with $|C'| \leq k$. Assume $C = \{C_i \mid i \in [1, n]\}$ and $|S| = |\bar{z}|$. Then each R_i encodes a subset $C_i \in C$, $y_i \in \bar{y}$ indicates C_i, and z_{i1}, z_{i2} and z_{i3} denote elements in C_i. Moreover, C contains a cover C' with $|C'| \leq k$ if and only if Q can be boundedly specialized with $\bar{x}$ from X and $|\bar{x}| \leq k$. This illustrates why QSP analysis is nontrivial. □

Theorem 5.3 gives the complexity of QSP. While $\mathsf{QSP}(\mathcal{L})$ has the same complexity as $\mathsf{UEP}(\mathcal{L})$, the proofs are quite different from their counterparts for UEP. Compared to LEP, the QSP analysis is more complicated for UCQ and $\exists\mathsf{FO}^+$.

Theorem 5.3: QSP is

- NP-complete for CQ; and
- Π_2^p-complete for UCQ and $\exists\mathsf{FO}^+$; and
- undecidable for FO. □

Proof sketch. (1) *Lower bounds.* We show that QSP is NP-hard, Π_2^p-hard and undecidable for CQ, UCQ and FO by reduction from MSC, $\forall^*\exists^*$3CNF and the complement of the satisfiability problem for FO, respectively. It is known that MSC is NP-complete (cf. [31]). In contrast to the reductions of Theorem 4.7, the reductions here encode what variables can be instantiated and ensure that all instantiations of these variables yield a covered specialized query. For instance, Example 5.2 outlines a reduction from MSC for CQ.

(2) *Upper bounds.* We develop NP and Π_2^p algorithms for checking QSP for CQ and $\exists\mathsf{FO}^+$, respectively. The algorithms make use of Theorem 3.11 and a lemma: if Q is $\mathcal{A}$-satisfiable, then for all tuples $\bar{x}$ of parameters of Q, there exists a valuation $\bar{c}$ of $\bar{x}$ such that $Q(\bar{x} = \bar{c})$ is $\mathcal{A}$-satisfiable. □

A syntactic condition. Is it possible to maintain an access schema $\mathcal{A}$ over a relational schema $\mathcal{R}$ such that bounded specialization is always within reach under $\mathcal{A}$ for all FO queries defined over $\mathcal{R}$? The answer is affirmative.

We say that $\mathcal{A}$ *covers* $\mathcal{R}$ if for each relation schema R in $\mathcal{R}$, there exists an access constraints $R(X \to (Y, N))$ in $\mathcal{A}$ such that for each attribute B of R, either $B \in X$ or $B \in Y$, *i.e.,* indices are built on B or for B. We say that an FO query Q is *fully parameterized* if its set X of parameters includes all variables in Q. These suffice for bounded specialization.

Proposition 5.4: Under an access schema $\mathcal{A}$ that covers a relational schema $\mathcal{R}$, all fully parameterized FO queries defined over $\mathcal{R}$ can be boundedly specialized. □

Generalization. The results of this section carry over to access constraints with non-constant cardinality.

Corollary 5.5: Theorem 5.3 and Proposition 5.4 also hold on access constraints of the form $R(X \to Y, s(\cdot))$. □

6. CONCLUSION

We have investigated how to query big data by leveraging bounded evaluability, to compute exact answers if possible, and approximate answers otherwise by means of envelopes and bounded query specialization. We have identified several problems associated with bounded evaluability, and provided their complexity and characterizations. The main complexity results are summarized in Table 1, annotated with their corresponding theorems.

Queries	BEP($\mathcal{L}$)	CQP($\mathcal{L}$)	UEP($\mathcal{L}$)	LEP($\mathcal{L}$)	QSP($\mathcal{L}$)
CQ	EXPSPACE-c (Th. 3.4)	PTIME (Th. 3.11)	NP-c (Th. 4.4)	NP-c (Th. 4.7)	NP-c (Th. 5.3)
UCQ	EXPSPACE-c (Cor. 3.7)	Π_2^p-c (Th. 3.14)	Π_2^p-c (Th. 4.4)	NP-c (Th. 4.7)	Π_2^p-c (Th. 5.3)
$\exists$FO$^+$	EXPSPACE-c (Cor. 3.7)	Π_2^p-c (Th. 3.14)	Π_2^p-c (Th. 4.4)	DP-c (Th. 4.7)	Π_2^p-c (Th. 5.3)
FO	undecidable [17]	not defined for FO	undecidable (Th. 4.4)	undecidable (Th. 4.7)	undecidable (Th. 5.3)

Table 1: Complexity for reasoning about bounded evaluability ($\mathcal{C}$-c indicates $\mathcal{C}$-complete)

This work suggests a strategy to answer queries on big data as follows. (1) We develop and maintain an access schema $\mathcal{A}$ for an application. (2) Given a dataset D that satisfies $\mathcal{A}$, for all queries Q posed over D, we first check whether Q is boundedly evaluable under $\mathcal{A}$ or covered by $\mathcal{A}$; if so, we compute exact answers $Q(D)$ by accessing a bounded amount of data; otherwise we compute approximate query answers, by using envelopes or by interacting with users to get a boundedly specialized query.

One topic for future work is identify an effective syntax for boundedly evaluable FO queries. Another topic is to study UEP and LEP under general access constraints. A third topic is to study, given a query Q in a language $\mathcal{L}$, whether Q has envelopes in another language $\mathcal{L}'$, *e.g.,* to find envelopes in CQ for an FO query. Finally, it is interesting to explore envelopes with approximation ratios measured in terms of precision and recall, instead of absolute approximation [14].

Acknowledgments. Fan is supported in part by NSFC 61133002, 973 Program 2012CB316200, Shenzhen Peacock Program 1105100030834361, Guangdong Innovative Research Team Program 2011D005, EPSRC EP/J015377/1 and EP/M025268/1, and a Google Faculty Research Award. Cao and Deng are supported in part by NSFC 61421003 and 973 Program 2014CB340302.

7. REFERENCES

[1] *http://data.gov.uk/dataset/road-accidents-safety-data.*

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

[3] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. I. Jordan, S. Madden, B. Mozafari, and I. Stoica. Knowing when you're wrong: building fast and reliable approximate query processing systems. In *SIGMOD*, 2014.

[4] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: Queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.

[5] M. Armbrust, K. Curtis, T. Kraska, A. Fox, M. J. Franklin, and D. A. Patterson. PIQL: Success-tolerant query processing in the cloud. *PVLDB*, 5(3), 2011.

[6] 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.

[7] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, 2003.

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

[9] P. Barceló, L. Libkin, and M. Romero. Efficient approximations of conjunctive queries. *SICOMP*, 43(3):1085–1130, 2014.

[10] P. Barceló, J. L. Reutter, and L. Libkin. Parameterized regular expressions and their languages. *TCS*, 474:21–45, 2013.

[11] Y. Cao, W. Fan, and R. Huang. Making pattern queries bounded in big graphs. In *ICDE*, 2015.

[12] Y. Cao, W. Fan, and W. Yu. Bounded conjunctive queries. *PVLDB*, 2014.

[13] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, 1977.

[14] S. Chaudhuri and P. G. Kolaitis. Can datalog be approximated? *JCSS*, 55(2):355–369, 1997.

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

[16] Facebook. Introducing Graph Search. *https://en-gb.facebook.com/about/graphsearch*, 2013.

[17] W. Fan, F. Geerts, and L. Libkin. On scale independence for querying big data. In *PODS*, 2014.

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

[19] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractability to polynomial time. *PVLDB*, 3(1):1161–1172, 2010.

[20] R. Fink and D. Olteanu. On the optimal approximation of queries using tractable propositional languages. In *ICDT*, 2011.

[21] M. N. Garofalakis and P. B. Gibbons. Wavelet synopses with error guarantees. In *SIGMOD*, 2004.

[22] G. Gottlob, S. T. Lee, G. Valiant, and P. Valiant. Size and treewidth bounds for conjunctive queries. *JACM*, 59(3), 2012.

[23] R. Haenni and N. Lehmann. Resource bounded and anytime approximation of belief function computations. *IJAR*, 31, 2002.

[24] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *VLDB*, 1999.

[25] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, 2009.

[26] M. P. Kato, T. Sakai, and K. Tanaka. Structured query suggestion for specialization and parallel movement: effect on search behaviors. In *WWW*, 2012.

[27] A. Klug. On conjunctive queries containing inequalities. *J. ACM*, 35(1):146–160, 1988.

[28] P. G. Kolaitis, D. L. Martin, and M. N. Thakur. On the complexity of the containment problem for conjunctive queries with built-in predicates. In *PODS*, 1998.

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

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

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

[32] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, 1980.

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

[34] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *JCSS*, 54(1), 1997.

[35] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD*, 1999.

[36] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3), 1996.