Bounded Conjunctive Queries

Yang Cao[1] Wenfei Fan[1] Wenyuan Yu[2]
[1]University of Edinburgh [2]Facebook Inc.
{Y.Cao-17@sms, wenfei@inf}.ed.ac.uk wyu@fb.com

Abstract

A query Q is said to be *effectively bounded* if for all datasets D, there exists a subset D_Q of D such that $Q(D) = Q(D_Q)$, and the size of D_Q and time for fetching D_Q are *independent of* the size of D. The need for studying such queries is evident, since it allows us to compute $Q(D)$ by accessing a bounded dataset D_Q, *regardless of* how big D is. This paper investigates effectively bounded conjunctive queries (SPC) under an access schema $\mathcal{A}$, which specifies indices and cardinality constraints commonly used. We provide characterizations (sufficient and necessary conditions) for determining whether an SPC query Q is effectively bounded under $\mathcal{A}$. We study several problems for deciding whether Q is bounded, and if not, for identifying a minimum set of parameters of Q to instantiate and make Q bounded. We show that these problems range from quadratic-time to NP-complete, and develop efficient (heuristic) algorithms for them. We also provide an algorithm that, given an effectively bounded SPC query Q and an access schema $\mathcal{A}$, generates a query plan for evaluating Q by accessing a bounded amount of data in any (possibly big) dataset. We experimentally verify that our algorithms substantially reduce the cost of query evaluation.

1. Introduction

Query answering is expensive. Consider the problem to decide, given a query Q, a dataset D and a tuple t, whether $t \in Q(D)$, *i.e.,* whether t is an answer to Q in D. This problem is NP-complete for conjunctive queries (*i.e.,* SPC, defined with selection, projection and Cartesian product operators); and it is undecidable for queries in relational algebra ($\mathcal{RA}$, cf. [6]). When D is big, the cost of computing $Q(D)$ is prohibitive. Indeed, even a linear-time query processing algorithm may take days on a dataset D of PB size (10^{15} bytes), and years when D is of EB size (10^{18} bytes) [20].

This motivates us to ask the following question: is it possible to compute $Q(D)$ by only accessing (visiting and fetching) a small subset D_Q of D? More specifically, we want to know whether a query Q has the following properties. For *all* datasets D, there exists a subset $D_Q \subset D$ such that

(a) $Q(D_Q) = Q(D)$,

(b) D_Q consists of no more than M tuples, and

(c) D_Q can be effectively identified by using access information, with a cost *independent of* $|D|$.

Here access information includes indices and cardinality constraints, specified as an access schema $\mathcal{A}$; and M is a bound determined by $\mathcal{A}$ and Q only. We say that Q is *effectively bounded under* $\mathcal{A}$ if it satisfies all the three conditions above, and *bounded* if it satisfies conditions (a) and (b) only.

If Q is effectively bounded, then we can find a bounded dataset D_Q and compute $Q(D)$ by using D_Q, *independent of* the size of possibly big D. Moreover, when D grows, the performance does not degrade. In other words, we can *reduce* big D to a "small" D_Q of a manageable size.

Many real-life queries are actually (effectively) bounded.

Example 1: Social networks, *e.g.,* Facebook, allow us to tag a photo and show who is in it. Such a tag is a link to the person "tagged". Consider the following.

(1) A query Q_0 is to find all photos from an album a_0 in which a person u_0 is tagged by one of her friends. The relations needed for answering Q_0 include the following:

- in_album(photo_id, album_id) for photo albums,
- friends(user_id, friend_id) for friends, and
- tagging(photo_id, tagger_id, taggee_id), indicating that taggee_id is tagged by tagger_id in photo_id.

We abbreviate these as in_album(pid_1, aid), friends(uid, fid) and tagging(pid_2, tid_1, tid_2), respectively.

Given these, Q_0 can be written as an SPC query as follows:

$$Q_0(\mathsf{pid}_1) = \pi_{\mathsf{pid}_1}\sigma_C\big(\mathsf{in\_album}(\mathsf{pid}_1, \mathsf{aid}) \times \mathsf{friends}(\mathsf{uid}, \mathsf{fid}) \times \mathsf{tagging}(\mathsf{pid}_2, \mathsf{tid}_1, \mathsf{tid}_2)\big),$$

where the selection condition C is given as

$$\mathsf{aid} = a_0 \wedge \mathsf{uid} = u_0 \wedge \mathsf{pid}_1 = \mathsf{pid}_2 \wedge \mathsf{tid}_1 = \mathsf{fid} \wedge \mathsf{tid}_2 = \mathsf{uid}.$$

Observe the following. (a) A dataset D_0 consisting of these relations is possibly big; for instance, Facebook has more than 1 billion users with 140 billion friend links [17]. (b) Query Q_0 is *not* bounded: we can add new photos to album a_0, new friends of u_0 to friend, or new tuples to tagging, and Q_0 has to check these tuples when D_0 grows.

However, social networks often impose limits (cardinality constraints) on D_0, *e.g.,* (a) each album includes at most 1000 photos, (b) each person may claim up to 5000 friends, and (c) each person in a photo can only be tagged once [18]. Moreover, indices can be built on in_album(aid), friends(uid), and tagging(pid_1, tid_2). As will be seen later, these indices and constraints make an *access schema* $\mathcal{A}_0$.

Under access schema $\mathcal{A}_0$, Q_0 is *effectively bounded*: we can compute $Q_0(D_0)$ by accessing at most 7000 tuples *no matter how large D_0 is*, as follows: (a) select a set T_1 of at most 1000 pid's from in_album with aid $= a_0$, by using the index on in_album(aid); (b) get a set T_2 of at most 5000 fid's from friends with user_id $= u_0$, using the index on friends(uid); (c) using $\mathsf{tid}_2 = u_0$ and pid_2's from T_1, fetch a set T_3 of at most 1000 (pid_2, tid_1) tuples from tagging via the index on tagging(pid_2, tid_2); and (d) compute a join T_4 of T_2 and T_3. Then $Q_0(D_0) = \pi_{\mathsf{photo\_id}}(T_4)$. This query plan visits at most 7000 tuples in total. Moreover, these tuples can be efficiently identified and retrieved by using the indices.

(2) Queries like Q_0 are routinely posed on social networks. Thus we want a query Q_1, which is the same as Q_0 except

that uid and aid are not constants, *i.e.,* values u_0 and a_0 are not given. Query Q_1 is *not* bounded even under $\mathcal{A}_0$.

However, Q_1 can be taken as a *parameterized query*, a template with parameters (uid, aid, fid, pid_2, tid_1, tid_2) such that some of them can be substituted with constants when Q_1 is executed. We identify a *minimum* subset X_P of parameters of Q_1, referred to as *dominating parameters*, such that when values of X_P are given, Q_1 is effectively bounded under $\mathcal{A}_0$. For instance, uid and aid make a set of dominating parameters: as shown above, when they are instantiated, the query on D_0 can be answered by accessing at most 7000 tuples. We can find X_P and suggest it to users for instantiation.

(3) As another example, consider an arbitrary *Boolean* SPC query Q_2 that, given an instance D of a relational schema $\mathcal{R}$, returns true if and only if $Q_2(D)$ is nonempty. It is known that Q_2 is bounded even in the absence of access schema [19]. More specifically, $Q_2(D)$ can be computed by accessing at most $|Q_2|$ amount of data no matter how big D is. Indeed, no matter $Q_2(D)$ is true or false, it needs a *witness* D_Q of size $|Q|$ such that $Q_2(D_Q) = Q_2(D)$. □

The idea of answering queries with a bounded dataset was first explored in [9–11], and was formalized in [19] (referred to as scale independence there). To make practical use of the idea, several questions have to be settled. Given a query Q and an access schema $\mathcal{A}$, can we determine whether Q is (effectively) bounded under $\mathcal{A}$? What is the complexity? If Q is not bounded, can we find a dominating-parameter set X_P of Q such that Q becomes effectively bounded under $\mathcal{A}$ when X_P is instantiated? Given a dataset D, how can we compute $Q(D)$ by efficiently fetching a bounded D_Q, by using access information in $\mathcal{A}$? These questions are *non-trivial*. It is known that it is *undecidable* to decide whether Q is bounded for Boolean $\mathcal{RA}$ queries [19]. The questions are open for SPC queries, which are widely used in practice.

Contributions. This paper answers these questions for SPC queries. The main results are as follows.

(1) We formulate bounded SPC queries (Section 2). Following [19], we use an access schema $\mathcal{A}$ to specify indices and cardinality constraints for databases of a relational schema $\mathcal{R}$. We revise the notions of scale independence studied in [19]. We say that an SPC query Q is bounded if for all instances D of $\mathcal{R}$, there exists a $D_Q \subset D$ such that $Q(D) = Q(D_Q)$, and the size of D_Q is *independent of the size of* D. If in addition, D_Q can be efficiently fetched by using $\mathcal{A}$, then Q is effectively bounded. We show that some queries are bounded but are *not* effectively bounded.

(2) We study the problems of determining boundedness and effective boundedness (Section 3). We provide a set of deduction rules to decide whether an SPC query Q is bounded under an access schema $\mathcal{A}$, and show that the rules provide *a sufficient and necessary condition* for the boundedness. We also provide a *characterization* of effectively bounded Q under $\mathcal{A}$. In contrast to $\mathcal{RA}$ queries [19], these results tell us that there are systematic methods to decide whether SPC queries are bounded or effectively bounded under $\mathcal{A}$.

(3) We study several problems in connection with the (effective) boundedness of SPC queries, establish their complexity, and develop algorithms for them (Section 4). Given an SPC query Q and an access schema $\mathcal{A}$, we study problems to decide (a) whether Q is bounded under $\mathcal{A}$, (b) whether Q is effectively bounded under $\mathcal{A}$, (c) if Q is not effectively bounded, whether there exists a set X_P of dominating parameters of Q to make Q effectively bounded under $\mathcal{A}$, and (d) if so, how to find a minimum set X_P? We show that these problems are in $O(|Q|(|\mathcal{A}| + |Q|))$-time, $O(|Q|(|\mathcal{A}| + |Q|))$-time, NP-complete and NPO-complete, respectively. We develop efficient (heuristic) algorithms for these problems.

(4) We give a PTIME (polynomial time) algorithm to generate query plans for answering effectively bounded SPC queries Q under $\mathcal{A}$ (Section 5). The query plans allow us to answer Q in any (possibly big) dataset D by accessing a subset D_Q of D. The evaluation scales with the size of D: the size $|D_Q|$ of D_Q is decided by $\mathcal{A}$ and Q only, and D_Q can be fetched by using indices in $\mathcal{A}$ in time *independent of* $|D|$. We also study the problem for identifying a minimum D_Q, and show that its decision problem is NP-complete.

(5) We experimentally verify the efficiency and effectiveness of our algorithms, using real-life and synthetic data (Section 6). We find that our algorithms are efficient: they take at most 2.1 seconds to decide whether Q is effectively bounded under $\mathcal{A}$, and to generate a query plan for Q, when Q is defined on a relational schema with 19 tables and 113 attributes, and $\mathcal{A}$ consists of 84 constraints. Moreover, our bounded query evaluation approach is effective: on a real-life dataset D of 21.4GB, our query plan only accesses 3800 tuples and gets answers in 9.3 seconds on average, while MySQL takes *longer than 14 hours*. That is, our approach is *3 orders of magnitude* faster than MySQL. The improvement is *more substantial* when D grows, since our approach accesses a bounded subset of D *no matter how large* D *is*!

These results suggest an approach to answering queries in big data D. Given an SPC query Q and an access schema $\mathcal{A}$, we first check in $O(|Q|(|\mathcal{A}| + |Q|))$-time whether Q is effectively bounded under $\mathcal{A}$. If so, we compute $Q(D)$ by accessing a bounded $D_Q \subset D$, *independent of* $|D|$. If not, we may either identify a minimum set of dominating parameters and invite users to supply their values, or suggest users to extend their access schema, such that Q becomes effectively bounded. Only when none of these is possible, we pay the price of computing $Q(D)$ directly in big D. As remarked earlier, many real-life queries are effectively bounded and can be processed regardless of the size of D, especially parameterized queries commonly used in, *e.g.,* recommender systems.

For all the results of the paper we give proof sketches in the paper; detailed proofs can be found in [5].

Related work. We characterize related work as follows.

Scale independence. The notion of boundedness is a revision of scale independence proposed in [10]. Scale independence aims 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 [9] 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 tuples are retrieved to meet the bounds, and hence lead to approximate answers. Scale independence was also studied in the presence of materialized views [11].

The notion of scale independence was recently formalized in [19]. The notion of access schema was also proposed there. For a given bound M, [19] defines a scale independent query Q to be one that for all datasets D, there exists $D_Q \subseteq D$ such that $Q(D) = Q(D_Q)$ and $|D_Q| \leq M$. It studies several decision problems for scale independence. In particular, it shows that it is undecidable to check whether a Boolean $\mathcal{RA}$ query is scale independent. It also develops a set of rules as a sufficient condition for deciding whether an $\mathcal{RA}$ query is scale independent under an access schema. No characterization was given there for scale independent SPC queries.

This works extends [19] as follows. (1) We do not require the size of D_Q to be bounded by a predefined M. Indeed, as long as $|D_Q|$ is determined by $\mathcal{A}$ and Q only, its evaluation scales well with D. Hence, we revise the notion of scale independence and define (effectively) bounded queries. (2) We provide *characterizations* for (effectively) bounded SPC queries Q under $\mathcal{A}$. As opposed to $\mathcal{RA}$ queries of [19], these give us *sufficient and necessary* conditions for deciding whether Q is (effectively) bounded. (3) We show that the (effective) boundedness of SPC queries can be decided in PTIME when M is not part of the input, but is NP-complete in the setting of [19] (when M is predefined), in contrast to the undecidability of the problem for $\mathcal{RA}$ queries. (4) None of the problems for dominating parameters was studied in [19]. (5) We give efficient (heuristic) algorithms for checking whether Q is (effectively) bounded, identifying dominating parameters, and for generating a query plan when Q is effectively bounded. No algorithms were provided in [19].

Making big data small. There have been several data reduction schemes that, given a dataset D, find a small dataset D' such that one can evaluate queries posed on D by using D' instead. These include compression, summarization and data synopses such as histograms, wavelets, quantile summaries, clustering and sampling [7, 13, 16, 21–24, 26, 29, 30]. Recently BlinkDB [8] has revised the idea to evaluate queries on big data. It adaptively samples data to find approximate query answers within a probabilistic error-bound and time constraints. Similar ideas were also explored in [9].

This work differs from the prior work as follows. (1) We aim to compute *exact answers* by using a bounded dataset whenever possible, rather than approximate query answers [8, 9]. (2) The prior reduction schemes [7, 13, 16, 21–24, 26, 29, 30] use *the same* dataset D' to answer *all queries* posed on D. In contrast, we adopt a *dynamic reduction scheme* that finds a small D_Q for each query Q. Here D_Q consists of only the information needed for answering Q and hence, allows us to compute $Q(D)$ by using a small dataset D_Q.

Access schema. Cardinality constraints have been studied for relational data (*e.g.,* [25]). Following [19], this paper aims to identify a bounded dataset D_Q to answer a query by making use of available indices and cardinality constraints.

As remarked in [19], access schema is quite different from access patterns [14, 15, 27]. Access patterns require that a relation can only be accessed by providing certain combinations of attribute values. In contrast, access schemas combine indexing and cardinality constraints, and guide us to find a bounded dataset D_Q for query answering. None of our results follows the previous work on access patterns.

2. Bounded Queries under an Access Schema

Below we first review SPC queries, and then present access schemas. Based on these, we define bounded and effectively bounded SPC queries under an access schema.

SPC. Consider a relational schema $\mathcal{R} = (R_1, \ldots, R_l)$ in which each R_i is a relation schema. Recall that an SPC query over $\mathcal{R}$ has the following form (see, *e.g.,* [6]):

$$Q(Z) = \pi_Z \sigma_C (S_1 \times \ldots \times S_n).$$

Here S_j is a (renaming of a) relation schema in $\mathcal{R}$, Z is a set of attributes of $\mathcal{R}$, and C is the *selection condition* of Q, defined as a conjunction of equality atoms $x = y$ or $x = c$. where x, y are attributes and c is a constant. We refer to attributes that appear in Z or C as *the parameters of* Q.

To simplify the discussion, we consider Q defined over a single schema $R(A_1, \ldots, A_m)$. This does not lose generality due to the lemma below, in which we denote by $\mathsf{inst}(\mathcal{R})$ the set of all database instances of relational schema $\mathcal{R}$.

Lemma 1: *For any relational schema $\mathcal{R}$, there exist a single relation schema R, a linear-time function g_D from $\mathsf{inst}(\mathcal{R})$ to $\mathsf{inst}(R)$, and a linear-time query-rewriting function g_Q from SPC to SPC such that for any instance D of $\mathcal{R}$ and any SPC query Q over $\mathcal{R}$, $Q(D) = g_Q(Q)(g_D(D))$.* □

Proof: The proof is simple: define R to be the schema consisting of all attributes of relation schemas of $\mathcal{R}$ (after proper renaming) and a new attribute A_R with $\mathsf{dom}(A_R) = [1, l]$. For each instance D of $\mathcal{R}$, $g_D(D)$ is the disjoint union of relations of D such that $t[A_R] = j$ for each tuple t of schema R_j, indicating its source. The rewriting function $g_Q(Q)$ simply replaces each occurrence R_j in Q with $\pi_{R_j}(\sigma_{A_R=j} R)$. □

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

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

where X and Y are sets of attributes of R, and N is a natural number. A database D of R *satisfies* the constraint if

- for any X-value $\bar{a}$, $|D_Y(X = \bar{a})| \leq N$, where $D_Y(X = \bar{a}) = \{t[Y] \mid t \in D, t[X] = \bar{a}\}$; that is, for each X value, there exist at most N distinct corresponding Y values;
- there exists an *index on X for Y* such that, given a X-value $\bar{a}$, it finds $D' \subseteq D$ such that $|D'| \leq N$ and $D'_Y(X = \bar{a}) = D_Y(X = \bar{a})$ with a cost measured in N.

Here D' is one of (possibly many) subsets of D with N tuples, one for each distinct value of Y, and N is independent of $|D|$. We say that D *satisfies* access schema $\mathcal{A}$, denoted by $D \models \mathcal{A}$, if D satisfies all the constraints in $\mathcal{A}$.

An access constraint is a combination of a cardinality constraint and an index. It tells us that for any given X-value. there exist a bounded number of corresponding Y values, and the Y values can be efficiently retrieved with the index.

Example 2: Recall from Example 1 the limit of 1000 photos per album. This can be expressed as an access constraint over schema in_album with an index on album_id for photo_id:

$$\mathsf{album\_id} \to (\mathsf{photo\_id}, 1000).$$

We enforce that each person is tagged at most once in a photo by an access constraint over tagging:

$$(\mathsf{photo\_id}, \mathsf{taggee\_id}) \to (\mathsf{tager\_id}, 1).$$

Similarly, the limit of 5000 friends per person can be expressed as $\mathsf{user\_id} \to (\mathsf{friend\_id}, 5000)$ over friends. □

Observe the following. (a) Functional dependencies (FDs) $X \to Y$ (see [6]) are a special case of access constraints of the form $X \to (Y, 1)$ if an index is defined on X for Y. (b) Keys are a special form of access constraints $X \to (R, 1)$, where R denotes all the attributes of relation schema R. In general, given an access constraint $X \to (R, N)$, we can efficiently fetch the entire tuples when an X value is given.

Bounded and effectively bounded SPC queries. We say that an SPC query Q over relation schema R is *bounded under* an access schema $\mathcal{A}$ if for *all* instances D of R that satisfies $\mathcal{A}$, there exists a subset $D_Q \subseteq D$ such that

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

(b) the size $|D_Q|$ is *independent of* the size $|D|$ of D.

Here $|D|$ is measured as the total *number of tuples* in D.

We say that Q is *effectively bounded under* $\mathcal{A}$ if Q is bounded under $\mathcal{A}$ and moreover, there exists an algorithm that identifies D_Q in time determined by Q and $\mathcal{A}$, *not* $|D|$.

Intuitively, Q is bounded under $\mathcal{A}$ if it can be answered in a bounded D_Q. It is effectively bounded if moreover, D_Q can be efficiently identified (assuming that given an X-value $\bar{a}$, it takes $O(N)$ time to identify $D_Y(X = \bar{a})$ in D via an access constraint $X \to (Y, N)$ in $\mathcal{A}$). For instance, as shown in Example 1, all Boolean SPC queries are bounded even in the absence of access schema, and query Q_0 is effectively bounded under the access schema $\mathcal{A}_0$ of Example 2.

The result below separates the class $\mathsf{SPC_b}$ of bounded queries from the class $\mathsf{SPC_{eb}}$ of effectively bounded queries under the same access schema, *i.e.,* $\mathsf{SPC_{eb}} \subset \mathsf{SPC_b}$.

Proposition 2: *There exists a query that is bounded but is not effectively bounded under the same access schema.* $\Box$

Proof: Consider a Boolean SPC query Q_2 that, given a dataset D, returns true iff there exists a tuple $t \in D$ such that $t[A] = c$ for a constant c. As argued in Example 1, Q_2 is bounded under an empty access schema $A_\emptyset = \emptyset$. However, Q_2 is not effectively bounded under $A_\emptyset$. Indeed, the lower bound for searching such a tuple t is $O(\log_2(|D|))$-time, even when D is sorted, which is not independent of $|D|$. $\Box$

3. Characterizing Effective Boundedness

We now provide sufficient and necessary conditions for determining the (effective) boundedness of SPC queries Q under an access schema $\mathcal{A}$. The main result of the section is as follows. (1) There exists a set $\mathcal{I}_B$ of deduction rules such that Q is bounded *if and only if* it can be proven from Q and $\mathcal{A}$ using $\mathcal{I}_B$. (2) Similarly, there exists a set $\mathcal{I}_E$ of such rules for effectively boundedness. These yield *characterizations* of (effective) boundedness via symbolic computation. Moreover, they reveal insight into the boundedness analysis, which helps us develop checking algorithms in Section 4.

We give $\mathcal{I}_B$ and $\mathcal{I}_E$ in Sections 3.1 and 3.2, respectively.

3.1 Deduction Rules for Boundedness

Consider an SPC query $Q(Z) = \pi_Z \sigma_C(S_1 \times \ldots \times S_n)$, where S_i is a renaming of relation schema R. We use Σ_Q to denote the set of all equality atoms $S[A] = S'[A']$ or $S[A] = c$ derived from the selection condition C of Q by the transitivity of equality. We use X and X' to denote sets of attributes of Q. We write $\Sigma_Q \vdash X = X'$ if $X = X'$ can be derived from equality atoms in Σ_Q, which can be checked in $O(\mathsf{max}(|X|, |X'|))$ time by leveraging a list of attributes in Q that can be precomputed in $O(|Q|^2)$ time.

(**Reflexivity**) If $X' \subseteq X$, then $X \mapsto_{\mathcal{I}_B} (X', 1)$.
(**Actualization**) If $X \to (Y, N)$ is in $\mathcal{A}$, then
$S_i[X] \mapsto_{\mathcal{I}_B} (S_i[Y], N)$ for each i in $[1, n]$.
(**Augmentation**) If $X \mapsto_{\mathcal{I}_B} (Y, N)$, then
$X \cup W \mapsto_{\mathcal{I}_B} (Y \cup W, N)$.
(**Transitivity**) If $X \mapsto_{\mathcal{I}_B} (Y_1, N_1)$, $Y_2 \mapsto_{\mathcal{I}_B} (W, N_2)$,
and $\Sigma_Q \vdash Y_1 = Y_2$, then $X \mapsto_{\mathcal{I}_B} (W, N_1 * N_2)$.

Figure 1: Deduction rules $\mathcal{I}_B$ for boundedness

To simplify the discussion we assume *w.l.o.g.* that attributes in S_i's have distinct names via renaming; see, *e.g.,* query Q_0 of Example 1. We also assume *w.l.o.g.* that Q is satisfiable, *i.e.,* Σ_Q does not includes $S[A] = c$ and $S[A] = d$ when c and d are distinct constants.

Rules. We present a set $\mathcal{I}_B$ of four deduction rules in Fig. 1. Given an SPC query Q and an access schema $\mathcal{A}$, we write

$$X \mapsto_{\mathcal{I}_B} (Y, N)$$

if $X \to (Y, N)$ can be deduced from $\mathcal{A}$ and Σ_Q by using the rules in $\mathcal{I}_B$. Here $X \mapsto_{\mathcal{I}_B} (Y, N)$ *extends* access constraints of Section 2 by allowing X and Y to be sets of attributes of Q from possibly multiple renamed relations of R in Q.

One can draw an analogy of $\mathcal{I}_B$ to our familiar Armstrong's Axioms for FD implication (see, *e.g.,* [6]).

(1) Reflexivity, Augmentation and Transitivity are immediate extensions of Armstrong's Axioms to access constraints. In particular, Transitivity allows us to propagate boundedness from one relation to another in a Cartesian product $S_1(X, Y_1) \times S_2(Y_2, W)$: if for any X-value $\bar{a}$, there exist at most N_1 distinct Y_1 values, then so do $S_2[Y_2]$ by $\Sigma_Q \vdash S_1[Y_1] = S_2[Y_2]$. Then from $Y_2 \to (W, N_2)$, it follows that given $\bar{a}$, there exist at most $N_1 * N_2$ distinct $S_2[W]$ values.

(2) Actualization is an application of some access constraint of $\mathcal{A}$ to a renaming S_i of R that appears in Q.

Example 3: Recall relation schemas $\mathsf{in\_album}(\mathsf{pid_1}, \mathsf{aid})$, $\mathsf{friends}(\mathsf{uid}, \mathsf{fid})$ and $\mathsf{tagging}(\mathsf{pid_2}, \mathsf{tid_1}, \mathsf{tid_2})$ given in Example 1. Let X_0 be (aid, uid, $\mathsf{tid_2}$, fid, $\mathsf{tid_1}$).

We show below how $X_0 \mapsto_{\mathcal{I}_B} (y, N_y)$ is proven from query Q_0 of Example 1 and access schema $\mathcal{A}_0$ of Example 2 by using $\mathcal{I}_B$, for each parameter y in Q_0 (*i.e.,* σ_C or Z) and for some positive integer N_y determined by Q_0 and $\mathcal{A}_0$.

(1) $\mathsf{aid} \mapsto_{\mathcal{I}_B} (\mathsf{pid_1}, 1000)$	Actualization
(2) $\mathsf{pid_2} \mapsto_{\mathcal{I}_B} (\mathsf{pid_2}, 1)$	Reflexivity
(3) $\Sigma_{Q_0} \vdash \mathsf{pid1} = \mathsf{pid2}$	selection condition in Q_0
(3) $\mathsf{aid} \mapsto_{\mathcal{I}_B} (\mathsf{pid_2}, 1000)$	by (1), (2), (3) and Transitivity
(4) $X_0 \mapsto_{\mathcal{I}_B} (\mathsf{aid}, 1)$	Reflexivity
(5) $X_0 \mapsto_{\mathcal{I}_B} (\mathsf{pid_2}, 1000)$	(3)(2) and Transitivity

Similarly, $X_0 \mapsto_{\mathcal{I}_B} (\mathsf{tid_1}, 1)$, $X_0 \mapsto_{\mathcal{I}_B} (\mathsf{tid_2}, 1)$, $X_0 \mapsto_{\mathcal{I}_B} (\mathsf{uid}, 1)$ and $X_0 \mapsto_{\mathcal{I}_B} (\mathsf{fid}, 1)$ by Reflexivity. $\Box$

Characterization. We next show that $\mathcal{I}_B$ provides *a sufficient and necessary condition* for determining whether an SPC query $Q(Z)$ is bounded under an access schema $\mathcal{A}$.

We use the following notations: (a) X_B is the set of all parameters of Q that appear in the selection condition σ_C such that for any $S[A] \in X_B$ and any $z \in Z$, $\Sigma_Q \not\vdash S[A] = z$, *i.e.,* attributes that involve in Boolean condition checking but are not part of the output; and (b) X_C is the set of all attributes such that for all $S[A] \in X_C$, $\Sigma_Q \vdash S[A] = c$ for some constant c, *i.e.,* already instantiated with constants.

Theorem 3: *An* SPC *query $Q(Z)$ is bounded under an access schema $\mathcal{A}$ if and only if for each parameter y in $X_B \cup Z$, $X_B \cup X_C \mapsto_{\mathcal{I}_B} (y, N_z)$, where N_z is a positive integer.* □

That is, Q is bounded under $\mathcal{A}$ iff for each "free variable" $z \in Z$ of Q, its boundedness can be deduced using $\mathcal{I}_B$ from (a) those parameters already instantiated in Q, and (b) those that only participate in condition checking and hence only need a witness for the truth value of the condition.

Proof: To verify this, we define a notion of *access closures*. Let X be a set of attributes in Q. The *access closure* X^* of X under $\mathcal{A}$ for Q is the set of all attributes y in Q such that for all $D \models \mathcal{A}$, there exists $D' \subseteq D$ such that (a) $Q(D) = Q(D')$, and (b) for all X values $\bar{a}$, $|\pi_y \sigma_{X=\bar{a}}(D)| \leq N_y$ for some positive integer N_y independent of $|D|$. Here $\sigma_{X=\bar{a}}(D)$ is short for $\sigma_{X_1=\bar{a}_1 \wedge \cdots \wedge X_n=\bar{a}_n}(S_1 \times \cdots \times S_n)(D)$, where for each $i \in [1, n]$, (i) X_i is the set of attributes in X that are from S_i; and (ii) $\bar{a}_i$ is the set of values in $\bar{a}$ for X_i.

It suffices to show the following lemmas: under $\mathcal{A}$,

- $Q(Z)$ is bounded *if and only if* $X_B \cup Z \subseteq (X_B \cup X_C)^*$;
- $X \mapsto_{\mathcal{I}_B} (Y, N)$ for some bound N *if and only if* $Y \subseteq X^*$, for any sets X and Y of attributes in Q.

For if these hold, $Q(Z)$ is bounded iff $X_B \cup Z \subseteq (X_B \cup X_C)^*$ iff $X_B \cup X_C \mapsto_{\mathcal{I}_B} (X_B \cup Z, N)$ iff $X_B \cup X_C \mapsto_{\mathcal{I}_B} (y, N_y)$ for each $y \in X_B \cup Z$. Hence Theorem 3 follows. □

Example 4: For query $Q_0(Z)$ given in Example 1, $Z = \{\mathsf{pid}_1\}$, $X_B = \{\mathsf{tid}_1, \mathsf{fid}\}$, and $X_C = \{\mathsf{uid}, \mathsf{aid}, \mathsf{tid}_2\}$. By the deduction of $\mathcal{I}_B$ given in Example 3, $X_B \cup X_C \mapsto_{\mathcal{I}_B} (\mathsf{pid}_1, 1000)$, $X_B \cup X_C \mapsto_{\mathcal{I}_B} (\mathsf{tid}_1, 1000)$, $X_B \cup X_C \mapsto_{\mathcal{I}_B} (\mathsf{fid}, 1)$. Hence Q_0 is bounded under $\mathcal{A}_0$ by Theorem 3.

Now consider an arbitrary Boolean SPC query $Q(Z)$ under access schema $\mathcal{A}_\emptyset = \emptyset$. The set Z of parameters for projection is $\emptyset$, and $X_B \mapsto_{\mathcal{I}_B} (x, 1)$ for any $x \in X_B$ by Reflexivity. Thus Q is bounded under $\mathcal{A}_\emptyset$ by Theorem 3. □

3.2 Deduction Rules for Effective Boundedness

To decide whether an SPC query $Q(Z)$ is effectively bounded under $\mathcal{A}$, more needs to be done. When we propagate the boundedness from a set X of attributes to another set Y, we have to ensure that the values of Y can be efficiently retrieved *via available indices* in $\mathcal{A}$. Below we develop a set $\mathcal{I}_E$ of deduction rules by incorporating this condition.

Rules. Consider an access schema $\mathcal{A}$ over schema R and a set Y_R of attributes of R. We say that Y_R is *indexed* in $\mathcal{A}$ if there exists $X_R \subseteq Y_R$ such that (1) $X_R \to (W, N)$ is an access constraint in $\mathcal{A}$; and (2) $Y_R \subseteq X_R \cup W$.

If Y_R is indexed, given a value $\bar{b}$, we can check whether $Y_R = \bar{b}$ is in a dataset $D \models \mathcal{A}$ by using indices in $\mathcal{A}$. Otherwise, we cannot decide this without searching the entire D. Thus the condition is *necessary* for effective boundedness.

Consider an SPC query $Q(Z) = \pi_Z \sigma_C(S_1 \times \ldots \times S_n)$ and a set $Y = (Y_1, \ldots, Y_n)$ of *parameters in* Q (*i.e.,* in C or Z), where Y_i consists of attributes from S_i. We say that Y is *indexed in* $\mathcal{A}$ if each Y_i is indexed in $\mathcal{A}$.

Using these, we give a set $\mathcal{I}_E$ of five rules for deducing the effective boundedness of SPC queries, in Fig. 2. We define $X \mapsto_{\mathcal{I}_E} (Y, N)$ along the same lines as $X \mapsto_{\mathcal{I}_B} (Y, N)$, using $\mathcal{I}_E$. While Reflexivity, Actualization and Transitivity of $\mathcal{I}_E$ are the same as their counterparts in $\mathcal{I}_B$, the others are not.

(1) Augmentation in $\mathcal{I}_E$ revises its counterpart in $\mathcal{I}_B$ by allowing Y to be extended with only indexed attributes.

(2) Combination also restricts Augmentation of $\mathcal{I}_B$ by enforcing the indexing condition; *i.e.,* for any X_i-value $\bar{a}_i$, if $\bar{a}_i$ is in $\pi_{X_i}(D)$ for a dataset $D \models \mathcal{A}$, then the deduced Y-value must be in $\pi_Y(D)$ and can be retrieved via indices. Note that Augmentation is a special case of Combination; we opt to keep Augmentation in $\mathcal{I}_E$ as it is easier to use.

(**Reflexivity**) If $X' \subseteq X$, then $X \mapsto_{\mathcal{I}_E} (X', 1)$.
(**Actualization**) If $X \to (Y, N)$ is in $\mathcal{A}$, then $S_i[X] \mapsto_{\mathcal{I}_E} (S_i[Y], N)$ for each i in $[1, n]$.
(**Transitivity**) If $X \mapsto_{\mathcal{I}_E} (Y, N)$ and $Y \mapsto_{\mathcal{I}_E} (W, N')$, then $X \mapsto_{\mathcal{I}_E} (W, N * N')$.
(**Augmentation**) If $X \mapsto_{\mathcal{I}_E} (Y, N)$ and $X \cup Y$ is *indexed*, then $X \mapsto_{\mathcal{I}_E} (X \cup Y, N)$.
(**Combination**) If $X_1 \mapsto_{\mathcal{I}_E} (Y_1, N_1)$, ..., $X_k \mapsto_{\mathcal{I}_E} (Y_k, N_k)$, $\Sigma_Q \vdash Y_1 = Y_1'$, ..., $\Sigma_Q \vdash Y_k = Y_k'$, then $X_1 \cup \cdots \cup X_k \mapsto_{\mathcal{I}_E} (Y_1' \cup \cdots \cup Y_k', N_1 * \cdots * N_k)$.

Figure 2: Rules $\mathcal{I}_E$ for effective boundedness

Characterization. Based on $\mathcal{I}_E$, we give *a sufficient and necessary condition* for effective boundedness. For an SPC query $Q(Z) = \pi_Z \sigma_C(S_1 \times \ldots \times S_n)$, we use the following notations: for all $i \in [1, n]$, (a) X_C^i is the set of all attributes of S_i already instantiated in Q, *i.e.,* $X_C^i = \{S_i[A] \in S_i \mid \Sigma_Q \vdash S_i[A] = c$ for a constant $c\}$, where S_i denotes the set of all attributes of S_i; (b) $X_C = X_C^1 \cup \cdots \cup X_C^n$; (c) X_Q^i denotes the set of all parameters of S_i that appear in either C or Z of Q; and (d) $\mathcal{X}^{\mathcal{A}}$ is the set of subsets $S_i[X']$ of attributes such that $X \to (Y, N)$ is in $\mathcal{A}$ and $X \subseteq X' \subseteq X \cup Y$, for all $i \in [1, n]$. The characterization is given as follows.

Theorem 4: *An* SPC *query $Q(Z)$ is effectively bounded under an access schema $\mathcal{A}$ if and only if for each $i \in [1, n]$,*

(1) $X_Q^i \subseteq W$ for some $W \in \mathcal{X}^{\mathcal{A}}$; and

(2) $X_C \mapsto_{\mathcal{I}_E} (X_Q^i, N_i)$ for some natural number N_i that is determined by Q and $\mathcal{A}$ only. □

That is, the instantiated attributes X_C^i can be checked using indices, along with those attributes that participate in either output or Boolean conditions of Q. We will use this characterization to generate query plans in Section 5.

Proof: To prove this, we define a notion of *effective access closures*. Let X be a set of attributes in Q. The effective access closure of X under $\mathcal{A}$ for $Q(Z)$, denoted by X^+, is the set consisting of all those subsets Y of attributes of Q, such that for any $D \models \mathcal{A}$, there exists $D' \subseteq D$ such that (a) $Q(D) = Q(D')$; (b) for any X value $\bar{a}$ *in* D, $|\pi_Y \sigma_{X=\bar{a}}(S_1 \times \cdots \times S_n)(D')| \leq N_Y$ for some natural number N_Y determined by $\mathcal{A}$ and Q; and (c) D' can be identified in time determined by $\mathcal{A}$ and Q only, independent of $|D|$.

If suffices to show the following: under $\mathcal{A}$, for all $i \in [1, n]$,

- $Q(Z)$ is effectively bounded *if and only if* $X_Q^i \subseteq W$ for some W in $\mathcal{X}^{\mathcal{A}}$ and $X_Q^i \in X_C^+$; and
- $X \mapsto_{\mathcal{I}_E} (Y, N)$ *if and only if* $Y \in X^+$, for any sets X and Y of attributes in Q and for some natural number N that is determined by Q and $\mathcal{A}$ only.

For if these hold, $Q(Z)$ is effectively bounded under $\mathcal{A}$ iff $X_Q^i \subseteq W$ for some $W \in \mathcal{X}^{\mathcal{A}}$ and $X_Q^i \in X_C^+$ iff $X_Q^i \subseteq W$ for some $W \in \mathcal{X}^{\mathcal{A}}$ and $X_C \mapsto_{\mathcal{I}_E} (X_Q^i, N_i)$ for all $i \in [1, n]$. □

Example 5: We show that $Q_0(\mathsf{pid}_1)$ of Example 1 is effectively bounded under access schema $\mathcal{A}_0$ of Example 2. Following Theorem 4, we show the following:

(1) $\mathsf{aid} \mapsto_{\mathcal{I}_E} (\mathsf{pid}_1, 1000)$	Actualization
(2) $\mathsf{aid} \mapsto_{\mathcal{I}_E} ((\mathsf{aid}, \mathsf{pid}_1), 1000)$	(1) and Augmentation
(3) $(\mathsf{aid}, \mathsf{uid}) \mapsto_{\mathcal{I}_E} (\mathsf{aid}, 1)$	Reflexivity
(4) $(\mathsf{aid}, \mathsf{uid}) \mapsto_{\mathcal{I}_E} ((\mathsf{aid}, \mathsf{pid}_1), 1000)$	(3)(2) and Transitivity
(5) $\mathsf{uid} \mapsto_{\mathcal{I}_E} (\mathsf{fid}, 5000)$	Actualization
(6) $\mathsf{uid} \mapsto_{\mathcal{I}_E} ((\mathsf{uid}, \mathsf{fid}), 5000)$	Augmentation
(7) $(\mathsf{aid}, \mathsf{uid}) \mapsto_{\mathcal{I}_E} (\mathsf{uid}, 1)$	Reflexivity
(8) $(\mathsf{aid}, \mathsf{uid}) \mapsto_{\mathcal{I}_E} ((\mathsf{uid}, \mathsf{fid}), 5000)$	(7)(6) and Transitivity
(9) $\mathsf{uid} \mapsto_{\mathcal{I}_E} (\mathsf{uid}, 1)$	Reflexivity
(10) $(\mathsf{aid}, \mathsf{uid}) \mapsto_{\mathcal{I}_E} ((\mathsf{pid}_2, \mathsf{tid}_2), 1000)$	(1)(9) and Combination
(11) $(\mathsf{pid}_2, \mathsf{tid}_2) \mapsto_{\mathcal{I}_E} (\mathsf{tid}_1, 1)$	Actualization
(12) $(\mathsf{pid}_2, \mathsf{tid}_2) \mapsto_{\mathcal{I}_E} ((\mathsf{pid}_2, \mathsf{tid}_1, \mathsf{tid}_2), 1)$	Augmentation
(13) $(\mathsf{aid}, \mathsf{uid}) \mapsto_{\mathcal{I}_E} ((\mathsf{pid}_2, \mathsf{tid}_1, \mathsf{tid}_2), 1000)$	(10)(12) and Transitivity

Then (a) condition (1) of Theorem 4 is satisfied since aid, uid and tid_2 are in subsets $\{\mathsf{aid}\}$, $\{\mathsf{udi}\}$ and $\{\mathsf{pid}_2, \mathsf{tid}_2\}$ of $\mathcal{X}^{\mathcal{A}_0}$, respectively. (b) Condition (2) is satisfied by deduction steps (4), (8) and (13) above, and as $(\mathsf{pid}_2, \mathsf{tid}_2)$ is indexed in $\mathcal{A}_0$. Thus Q_0 is effectively bounded under $\mathcal{A}_0$ by Theorem 4. □

4. Boundedness: Complexity and Algorithms

We next study two issues in connection with the (effective) boundedness of SPC queries. (1) We study the complexity and algorithms for deciding whether an SPC query is (effectively) bounded under an access schema $\mathcal{A}$. (2) When Q is not effectively bounded, we study whether Q can be made effectively bounded under $\mathcal{A}$ by instantiating a set X_P of parameters of Q, and if so, how to compute a minimum X_P.

The main results of this section are as follows. (1) The boundedness of Q under $\mathcal{A}$ can be decided in quadratic time (Section 4.1). (2) The same complexity holds for effective boundedness (Section 4.2). (3) The decision problem for dominating parameters is NP-complete, and its optimization problem is NPO-complete. We provide an efficient heuristic algorithm to compute dominating parameters (Section 4.3).

4.1 Checking Boundedness

We start with the *boundedness problem* $\mathsf{Bnd}(Q, \mathcal{A})$:

- Input: A relation schema R, an SPC query Q over R, and an access schema $\mathcal{A}$ over R.
- Question: Is Q bounded under $\mathcal{A}$?

This is to decide whether for all datasets D that satisfy $\mathcal{A}$, there exists *at all* a subset D_Q such that $Q(D) = Q(D_Q)$ and $|D_Q|$ is independent of the size $|D|$ of the underlying D.

While this problem is undecidable for (Boolean) $\mathcal{RA}$ queries [19], it is decidable in PTIME for SPC.

Theorem 5: *For any* SPC *query Q and access schema $\mathcal{A}$,* $\mathsf{Bnd}(Q, \mathcal{A})$ *can be decided in $O(|Q|(|\mathcal{A}| + |Q|))$ time.* □

Here $|\mathcal{A}|$ and $|Q|$ are the size of $\mathcal{A}$ and Q, respectively, and are typically small in practice, compared to datasets D.

As a constructive proof for Theorem 5, we next give such an algorithm for checking the boundedness of Q under $\mathcal{A}$.

Algorithm BCheck. The algorithm is denoted by BCheck and shown in Fig. 3. It is based on the characterization

Algorithm BCheck

Input: An SPC query Q, and an access schema $\mathcal{A}$.
Output: "yes" if Q is bounded under $\mathcal{A}$ and "no" otherwise.

```
1.  Γ := Actualize(A, Q);    /*Initialization*/
2.  closure := X_B ∪ X_C; B := X_B ∪ X_C;
3.  for each attribute A in A and Q and each φ in Γ do
4.      if isIn(φ, A, Q) then /*suppose that φ is X_φ ↦_{I_B} (Y_φ, N_φ)*/
5.          add φ to L[A]; n_φ := |X_φ|;
6.  while B is not empty do    /*Computation*/
7.      A := B.pop();
8.      for each φ in L[A] do
9.          decrease n_φ with 1;
10.         if n_φ = 0 do /*suppose that φ is X_0 ↦_{I_B} (Y_0, N)*/
11.             B := B ∪ (Y_0 \ closure);
12.             for each attribute B_0 in Y_0 do
13.                 for all B'_0 such that Σ_Q ⊢ B_0 = B'_0 do
14.                     add B'_0 to closure;
15. if X_B ∪ Z ⊆ closure then return "yes";   /*Checking*/
16. return "no";
```

Figure 3: Algorithm BCheck

of $\mathcal{I}_B$ (Section 3). It computes $(X_B \cup X_C)^*$, stored in a variable *closure*, and concludes that Q is bounded under $\mathcal{A}$ if and only if $X_B \cup Z \subseteq$ *closure*, *i.e.,* when all parameters of Q are covered by $(X_B \cup X_C)^*$ (see Theorem 3 and its proof).

More specifically, BCheck first actualizes access constraints of $\mathcal{A}$ in each renaming S_i of schema R in Q: for each $X \to (Y, N)$ in $\mathcal{A}$ and each S_i in Q, it includes $S_i[X] \mapsto_{\mathcal{I}_B} (S_i[Y], N)$ in a set Γ (line 1). Using Γ, it then computes *closure* (lines 2-14) such that if $X_B \cup X_C \mapsto_{\mathcal{I}_B} (y, N)$ for some N and attribute y, then y is included in *closure*. After this, it simply checks whether $X_B \cup Z$ is contained in *closure*; it returns "yes" if so and "no" otherwise (lines 15-16).

We next show how BCheck computes *closure*, starting with auxiliary structures used by BCheck.

Auxiliary structures. BCheck uses three auxiliary structures.

(1) BCheck maintains a set $\mathcal{B}$ of attributes in $\mathcal{A}$ and Q that are in *closure* but it remains to be checked what other attributes can be deduced from them via $\mathcal{I}_B$. Initially, $\mathcal{B} = X_B \cup X_C$ (line 2). BCheck uses $\mathcal{B}$ to control the **while** loop (lines 6–14): it terminates when $\mathcal{B} = \emptyset$, *i.e.,* when all necessary deduction checking via $\mathcal{I}_B$ has been completed.

(2) For each constraint ϕ: $X \mapsto_{\mathcal{I}_B} (Y, N)$ in Γ, BCheck maintains a counter n_ϕ to keep track of those attributes of X that are still in $\mathcal{B}$. Initially, n_ϕ is the number of attributes in X. When $n_\phi = 0$, *i.e.,* after all X attributes have been processed, the Y attributes can be added to $\mathcal{B}$ (lines 10-11).

(3) For each attribute A in Q and Γ, BCheck uses a list $L[A]$ to store all constraints $X \mapsto_{\mathcal{I}_B} (Y, N)$ in Γ such that either A is in X or there exists A' in X with $\Sigma_Q \vdash A = A'$. That is, $L[A]$ indexes constraints that are "applicable" to A.

Computing closure. With these structures, BCheck computes *closure* as follows. It first initializes the auxiliary structures as described above (lines 2-5). Here function $\mathsf{isIn}(\phi, A, Q)$ checks whether constraint ϕ: $X_\phi \mapsto_{\mathcal{I}_B} (Y_\phi, N_\phi)$ is "applicable" to attribute A, *i.e.,* whether there exists A' such that $\Sigma_Q \vdash A = A'$ and A' is in X_ϕ (line 5).

After this, BCheck processes attributes in $\mathcal{B}$ one by one (lines 6-14). For each attribute $A \in \mathcal{B}$ and each constraint $\phi : X_0 \mapsto_{\mathcal{I}_B} (Y_0, N)$ in $L[A]$, it decreases the counter n_ϕ by 1. When $n_\phi = 0$, *i.e.,* all attributes in X_0 have been inspected, BCheck conducts deduction via $\mathcal{I}_B$ (lines 11-14).

It adds to $\mathcal{B}$ attributes B_0 in Y_0 that are not yet in *closure* (line 11), and add to *closure* all those attributes B_0' such that $\Sigma_Q \vdash B_0 = B_0'$ (lines 12–14). When $\mathcal{B}$ becomes empty, BCheck returns "yes" iff $X_B \cup Z \subseteq$ *closure* (lines 15-16).

Correctness & Complexity. The correctness of BCheck follows from Theorem 3. To see that BCheck is in $O(|Q|(|\mathcal{A}|+|Q|))$ time, observe the following. (1) The initialization steps take $O(|Q||\mathcal{A}|)$ time (lines 1-5). (2) The *closure* is computed in $O(|\mathcal{A}| + |\mathcal{A}||Q|)$ time (lines 6-14), since the counters are updated at most $O(|\mathcal{A}||Q|)$ times *in total*, and each ϕ in Γ is used at most once, in $O(|\phi|+|Q|)$ time (thus $O(|\mathcal{A}|+|Q|)$ time in total). (3) The checking (line 15) can be done in $O(|Q|^2)$ time, since the size of *closure* is bounded by $O(|Q|)$.

Example 6: We show how algorithm BCheck finds that query Q_0 of Example 1 is bounded under the access schema $\mathcal{A}_0$ of Example 2. Here $X_B \cup X_C = \{\mathsf{aid}, \mathsf{uid}, \mathsf{tid}_2, \mathsf{fid}, \mathsf{tid}_1\}$. BCheck initializes Γ with $\mathsf{aid} \mapsto_{\mathcal{I}_B} (\mathsf{pid}_1, 1000)$ (ϕ_1), $(\mathsf{pid}_2, \mathsf{tid}_2) \mapsto_{\mathcal{I}_B} (\mathsf{tid}_1, 1)$ (ϕ_2), and $\mathsf{uid} \mapsto_{\mathcal{I}_B} (\mathsf{fid}, 5000)$ (ϕ_3). It assigns $X_B \cup X_C$ as the initial value of *closure* and $\mathcal{B}$, and sets counters $n_{\phi_1} = n_{\phi_3} = 1$, $n_{\phi_2} = 2$. After aid is popped off from $\mathcal{B}$, n_{ϕ_1} is decreased to 0 and BCheck updates *closure* and $\mathcal{B}$ with ϕ_1 (lines 11-14). Since $\Sigma_Q \vdash \mathsf{pid}_1 = \mathsf{pid}_2$, both pid_1 and pid_2 are added to *closure*, and pid_1 is added to $\mathcal{B}$. After this iteration, *closure* remains unchanged and $\mathcal{B}$ will be reduced to empty. Since $X_B \cup Z = \{\mathsf{pid}_1, \mathsf{pid}_2, \mathsf{tid}_1, \mathsf{fid}\}$ is a subset of *closure*, BCheck returns "yes". □

4.2 Checking Effective Boundedness

We next study the *effective boundedness problem*, denoted by $\mathsf{EBnd}(Q, \mathcal{A})$ and stated as follows:

- Input: R, Q and $\mathcal{A}$ as in $\mathsf{Bnd}(Q, \mathcal{A})$.
- Question: Is Q effectively bounded under $\mathcal{A}$?

It is to decide whether for any D that satisfies $\mathcal{A}$, we can fetch $D_Q \subseteq D$ via indices in $\mathcal{A}$ such that $Q(D) = Q(D_Q)$.

Problem EBnd is also decidable in quadratic-time.

Theorem 6: $\mathsf{EBnd}(Q, \mathcal{A})$ *is in* $O(|Q|(|\mathcal{A}| + |Q|))$ *time.* □

We prove Theorem 6 by providing an algorithm for checking the effective boundedness of Q under $\mathcal{A}$.

Algorithm EBCheck. The algorithm, denoted by EBCheck, extends algorithm BCheck by leveraging Theorem 4 and the following connection between $\mathcal{I}_E$ and the *access closure* for boundedness: for any sets X and Y of attributes in Q such that $X \subseteq Y$, $X \mapsto_{\mathcal{I}_E} Y$ if and only if $Y \subseteq X^*$ and Y is indexed in $\mathcal{A}$. Based on this, EBCheck works as follows.

Step 1 (computing closure): Compute X_C^* by adopting the *closure* computation part of BCheck (lines 1-14, Fig. 3) except that it initializes *closure* to be X_C instead of $X_B \cup X_C$.

Step 2 (checking): Check (a) whether $\bigcup_{i=1}^n X_Q^i$ is a subset of X_C^* and (b) whether $\bigcup_{i=1}^n X_Q^i$ is indexed in $\mathcal{A}$. If so, Q is effectively bounded under $\mathcal{A}$. Note that the condition (1) of Theorem 4 is implied by (b) here.

As both steps are in $O(|Q|(|\mathcal{A}|+|Q|)$ time, so is EBCheck.

Example 7: Consider again query Q_0 of Example 1 and access schema $\mathcal{A}_0$ of Example 2. The deduction analysis of Example 5 tells us that X_C^* of Q_0 covers parameters of in_album, friends and tagging; moreover, X_C^* is indexed by $\mathcal{A}_0$. That is, the conditions in Step 2 of EBCheck are satisfied. Hence, Q_0 is effectively bounded under $\mathcal{A}_0$. □

4.3 Computing Dominating Parameters

As illustrated in Example 1, when an SPC query Q is not effectively bounded under $\mathcal{A}$, we want to identify a minimum set X_P of parameters of Q such that if X_P is instantiated, Q becomes effectively bounded. We want to find and suggest such an X_P to users if it exists. When the users provide a value of X_P, Q can be answered in a big dataset D by accessing a bounded amount of data. We consider parameters of X_P that are not in X_C, *i.e.,* not yet instantiated in Q.

More specifically, we use $Q(X_P = \bar{a})$ to denote the query obtained from Q when X_P is given a value $\bar{a}$. We call X_P a set of *dominating parameters* of Q under $\mathcal{A}$ if $Q(X_P = \bar{a})$ is effectively bounded under $\mathcal{A}$ *for all* given X_P values $\bar{a}$.

Problems and complexity. This suggests that we study the following decision and optimization problems.

The dominating parameter problem $\mathsf{DP}(Q, \mathcal{A})$.

- Input: R, $Q(Z)$ and $\mathcal{A}$ as in $\mathsf{EBnd}(Q, \mathcal{A})$.
- Question: Does there exist a set of dominating parameters of Q under $\mathcal{A}$?

The minimum dominating parameter problem $\mathsf{MDP}(Q, \mathcal{A})$.

- Input: R, $Q(Z)$ and $\mathcal{A}$ as in $\mathsf{EBnd}(Q, \mathcal{A})$.
- Output: A minimum set of dominating parameters X_P of Q under $\mathcal{A}$, if it exists.

Problem $\mathsf{DP}(Q, \mathcal{A})$ is to decide whether Q has a set of dominating parameters at all. Problem $\mathsf{MDP}(Q, \mathcal{A})$ is to compute a minimum set of dominating parameters of Q.

Example 8: An SPC query may not have a set of dominating parameters under an access schema. As an example, consider query Q_0 of Example 1 and an access schema $\mathcal{A}_1$ that contains all access constraints in A_0 of Example 2 except $(\mathsf{photo\_id}, \mathsf{taggee\_id}) \rightarrow (\mathsf{tagger\_id}, 1)$. Then Q_0 is not effectively bounded under $\mathcal{A}_1$, and worse still, no matter what parameters of Q_0 we instantiate, it is still not effectively bounded. This is because no index is built on tagging in $\mathcal{A}_1$, and hence we cannot verify, *e.g.,* whether $\mathsf{tid}_2 = u_0$ is in a tagging instance without searching the entire D. □

While DP and MDP are important, they are hard.

Theorem 7: *For* SPC *query* Q *and access schema* $\mathcal{A}$,

(1) $\mathsf{DP}(Q, \mathcal{A})$ *is* NP*-complete; and*

(2) $\mathsf{MDP}(Q, \mathcal{A})$ *is* NPO*-complete.* □

NPO is the *class* of all NP optimization problems. NPO-complete problems are the hardest optimization problems in NPO: they do not even allow PTIME approximation algorithms with an exponential approximation ratio (cf. [12]).

Proof: We show that $\mathsf{DP}(Q, \mathcal{A})$ is NP-hard by reduction 3SAT. We verify its upper bound by giving an NP algorithm: guess a set X of parameters of Q and a proof started from X with bounded length $(O(|Q|(|Q|+|\mathcal{A}|)))$ from Q and $\mathcal{A}$ using $\mathcal{I}_E$, and then check whether (a) X is indexed in $\mathcal{A}$ and (b) the proof leads to $X \mapsto_{\mathcal{I}_E} (X_Q, N)$ for some N, in PTIME. The correctness of the algorithm follows from Theorem 4.

One can easily verify that the decision problem of MDP is in NP; hence MDP is in NPO. The NPO-hardness is verified by a PTAS-reduction from the MINIMUM WEIGHTED 3SAT problem, which is NPO-complete (cf. [12]). □

Algorithm. In light of Theorem 7, we develop a heuristic algorithm that, given Q and $\mathcal{A}$, checks whether there exists a set of dominating parameters for Q under $\mathcal{A}$; it finds and returns such a set X_P if so, and returns "no" otherwise. The algorithm, denoted by $\mathsf{findDP_h}$, consists of three steps.

Step 1 (initial candidates): For each renaming S_i of R in Q and each parameter A of Q that is in S_i but is not in X_C, add A to a set X_P if there exists a constraint $X \to (Y, N)$ in $\mathcal{A}$ such that A is in $S_i[X] \cup S_i[Y]$.

Step 2 (checking): Check (a) whether $\bigcup_{i=1}^{n} X_Q^i$ is indexed in $\mathcal{A}$ and (b) whether for all X_Q^i, $X_Q^i \subseteq X_P$ (see Section 3.2 for the definition of X_Q^i). If not, return "no".

Step 3 (minimizing): We optimize X_P iteratively as follows. Each time we pick one attribute A of some S_i from X_P, and check whether there exists $X \to (Y, N)$ in $\mathcal{A}$ such that $S_i[X] \subseteq X_P$, $A \notin S_i[X]$ and $A \in S_i[Y]$. If so, let $X_P = X_P \setminus \mathsf{ext_Q}(A)$ since X_P can be recovered from $X_P \setminus \{A\}$ via deduction of $\mathcal{I}_E$, where $\mathsf{ext_Q}(A)$ consists of all parameters x such that $\Sigma_Q \vdash A = x$. We then process the next attribute. We return X_P when it cannot be further reduced.

Correctness & Complexity. One can verify that if $\mathsf{findDP_h}$ returns X_P, then X_P is a set of dominating parameters for Q under $\mathcal{A}$. Indeed, if X_P is instantiated, then for all S_i in Q, all parameters in X_Q^i can be deduced from X_P via $\mathcal{I}_E$ and are also indexed. Hence $Q(X_P = \bar{a})$ is effectively bounded under $\mathcal{A}$ by Theorem 4, for any X_P value $\bar{a}$.

Algorithm $\mathsf{findDP_h}$ is in $O(|Q|(|Q| + |\mathcal{A}|))$ time. Indeed, its step 1 is in $O(|\mathcal{A}||Q|)$ time; step 2 takes $O(|Q|^2)$ time since $|X_P|$ and $|X_Q|$ are both bounded by $|Q|$; and step 3 is in $O(|Q|(|\mathcal{A}| + |Q|))$ time because $|X_P| \leq |Q|$; hence it takes $O(|\mathcal{A}|)$ time to check whether an attribute A can be removed from X_P, and $O(|Q|)$ time to remove $\mathsf{ext_Q}(A)$.

Example 9: Recall that query Q_1 of Example 1 is not effectively bounded under access schema $\mathcal{A}_0$ of Example 2. We show how $\mathsf{findDP_h}$ finds a set X_P of dominating parameters for Q_1. In step 1, it sets $X_P = \{\mathsf{pid_1}, \mathsf{aid}, \mathsf{uid}, \mathsf{fid}, \mathsf{pid_2}, \mathsf{tid_1}, \mathsf{tid_2}\}$. In step 2, $\mathsf{findDP_h}$ finds X_Q^i contained in X_P for X_Q^i in $\mathsf{in\_album}$, $\mathsf{friends}$ and $\mathsf{tagging}$; hence there exists a set of dominating parameters for Q_1. In step 3, it reduces X_P. (a) It first finds that $\mathsf{album\_id} \to (\mathsf{photo\_id}, 1000)$ in $\mathcal{A}_0$, and removes $\mathsf{pid_1}$ and $\mathsf{pid_2}$ from X_P since $\Sigma_Q \vdash \mathsf{pid_1} = \mathsf{pid_2}$. (b) It then finds $\mathsf{user\_id} \to (\mathsf{friend\_id}, 5000)$ in $\mathcal{A}_0$, and removes fid and $\mathsf{tid_1}$ from X_P by $\Sigma_Q \vdash \mathsf{fid} = \mathsf{tid_1}$. After this, $\mathsf{findDP_h}$ finds that it can remove no more parameters from X_P, and thus returns $X_P = \{\mathsf{aid}, \mathsf{uid}, \mathsf{tid_2}\}$, which is exactly the set of instantiated parameters for Q_0 (by $\Sigma_{Q_0} \vdash \mathsf{tid_2} = \mathsf{u_0}$). □

5. Algorithm for Effectively Bounded Queries

Algorithm $\mathsf{EBCheck}$ of Section 4.2 is able to determine the effective boundedness of SPC queries. However, it does not tell us *how to* identify a bounded amount of data to answer those queries. To bridge the gap, we next develop an algorithm that, given an effectively bounded SPC query $Q(Z) = \pi_Z \sigma_C (S_1 \times \ldots \times S_n)$ and an access schema $\mathcal{A}$, finds a query plan that, given a (big) dataset D, fetches a bounded $D_Q \subseteq D$ using indices in $\mathcal{A}$ such that $Q(D) = Q(D_Q)$.

The main results of the section are as follows. (1) There exists an $O(|Q|^2|\mathcal{A}|^3)$-time algorithm that generates query plans for effectively bounded SPC queries (Section 5.1). (2) We also study the problem to find a minimum bounded D_Q, and show that the problem is NP-complete (Section 5.2).

5.1 Determining and Computing D_Q

We find a query plan for Q by deducing a proof ρ_i for $X_C \mapsto_{\mathcal{I}_E} (X_Q^i, M_i)$ for all $i \in [1, n]$, following Theorem 4. Below we show that the proofs yield a query plan that, for any dataset D such that $D \models \mathcal{A}$, tells us how to find D_Q such that $Q(D) = Q(D_Q)$ and D_Q has at most $\sum_{i=1}^{n} M_i$ tuples.

Query plan from proofs. Suppose that $X_C \mapsto_{\mathcal{I}_E} (X_Q^i, M_i)$ is proven by $\rho_i = \varphi_1, \cdots, \varphi_m$, where φ_j denotes application of a rule in $\mathcal{I}_E$. We show that given D, ρ_i tells us how to find a list of subsets $T_1, \ldots, T_m$ of D such that

- $D_Q^i = \bigcup_{j=1}^{m} T_j$ and $D_Q = \bigcup_{i=1}^{n} D_Q^i$, and
- for all $j \in [1, m]$, $T_j \subseteq D$, T_j has at most N_j tuples and can be fetched by using indices in $\mathcal{A}$, where N_j is a number deduced from the proof, independent of $|D|$.

We can then compute $Q(D)$ by conducing joins and projections on these T_j's only, guided by conditions in σ_C of Q, as illustrated by how we get $Q_0(D_0)$ using T_1–T_4 in Example 1.

Below we show how to fetch T_j from D guided by rule φ_j, by giving two example rules (see [5] for other rules). Initially, $T_1 = \bigcup_{j=1}^{n} \sigma_{X_j = C_j}(D)$, and can be fetched by using indices in $\mathcal{A}$ on the constants of X_C (see Theorem 4 and its proof).

(a) When φ_j actualizes a constraint $X \to (Y, N)$ of $\mathcal{A}$, we fetch N tuples for T_j either from D by using index in $\mathcal{A}$ on X for Y, or from a bounded subset $T_{j'}$ of D ($j' < j$) deduced from previous steps in proof ρ_i, on which φ_j is applied.

(b) When φ_j is Combination, we get T_j as follows. Denote $\bigcup_{s=1}^{j-1} T_s$ by T. As indicated by the rule (Fig. 2), for $l \in [1, k]$, (i) all X_l and Y_l values are already fetched in T; and (ii) we can check whether these X_l and Y_l values appear in tuples of D, *i.e.,* they are contained in the projection of D on $\bigcup_{l=1}^{k} X_l \cup Y_l'$, by using the indices on the attributes. There are at most $N_1 * \ldots * N_k$ such tuples from T to be inspected in D, and T_j consists of these tuples.

Algorithm QPlan. We now present the algorithm, denoted by QPlan and shown in Fig. 4. Based on the connection between $\mathcal{I}_E$ proofs and query plans given above, QPlan focuses on finding a proof ρ_i for each $X_C \mapsto_{\mathcal{I}_E} (X_Q^i, M_i)$, based on the characterization of $\mathcal{I}_E$ of Section 3. It represents ρ_i as an object o_i, which consists of three components:

- $o_i.X$: parameters of X_Q^i deduced from the proof;
- $o_i.\mathcal{P}$: a proof for deducing $o_i.X$ from X_C; and
- $o_i.c$: the number of tuples that need to be fetched and inspected based on the query plan $o_i.\mathcal{P}$.

When o_i is completed, $o_i.\mathcal{P} = \rho_i$ and $o_i.c = M_i$.

Given an SPC query $Q(Z) = \pi_Z \sigma_C (S_1 \times \ldots \times S_n)$ that is effectively bounded under $\mathcal{A}$, QPlan returns a set $X_C^{\mathsf{min+}}$ of objects such that for $i \in [1, n]$, there exists $o_i \in X_C^{\mathsf{min+}}$ representing a proof for $X_C \mapsto_{\mathcal{I}_E} (X_Q^i, M_i)$.

More specifically, $X_C^{\mathsf{min+}}$ is a set of objects such that that Q is effective bounded under $\mathcal{A}$ if and only if for each $i \in [1, n]$, (1) $X_C^i \subseteq W$ for some W in $\mathcal{X}^{\mathcal{A}}$; and (2) $X_Q^i \subseteq o.X$ for some object o in $X_C^{\mathsf{min+}}$. It has a *coverage property*: for all Y, if $X \mapsto_{\mathcal{I}_E} (Y, N)$ and $X \subseteq Y$, then there exists some $o \in X_C^{\mathsf{min+}}$ such that $Y \subseteq o.X$. These suffice by Theorem 4.

We use the following notations. (a) A set S_2 of objects can

Algorithm QPlan

Input: An SPC query Q, and an access schema $\mathcal{A}$.
Output: A set $X_C^{\mathsf{min}+}$ of objects representing a query plan.

1. $X_C^{\mathsf{min}+} := \{o_C\}$; $\mathcal{B} := X_C^{\mathsf{min}+}$;
   /*$o_C.X = X_C$, $o_C.\mathcal{P} = \emptyset$, $o_c.c = 0$*/
2. $\Gamma :=$ Actualize$(\mathcal{A}, Q)$; $\mathcal{T} := nil$; /*Initialization*/
3. **while** $\mathcal{B}$ is not empty **do** /*Computing set $X_C^{\mathsf{min}+}$*/
4. $\quad o := \mathcal{B}$.pop();
5. $\quad$ **for each** $\phi : W \mapsto_{\mathcal{I}_E} (Y, N)$ in Γ and $W \subseteq o.X$ **do**
6. $\qquad$ instantiate o_Y for *possibly* deducing $o.X \cup Y$ from X_C;
7. $\qquad$ add o_Y to sets $\mathcal{T}$; remove ϕ from Γ;
8. $\quad$ **for each** $\Sigma_Q \vdash W = X'$, $X' \subseteq o.X$ and $W' \not\subseteq o.X$ **do**
9. $\qquad$ **if** $o.X \cup W \not\subseteq o'.X$ for any o' in $X_C^{\mathsf{min}+}$ **do**
10. $\qquad\quad$ instantiate o_W for *possibly* deducing $o.X \cup W$ from X_C;
11. $\qquad\quad$ add o_W to $\mathcal{T}$ for checking the *indexing* condition of γ_5;
12. $\quad U :=$ chkComb$(\mathcal{T}, X_C^{\mathsf{min}+})$; /*Deduce with Combination*/
13. $\quad \mathcal{B} := \mathcal{B} \cup U$; $X_C^{\mathsf{min}+} := X_C^{\mathsf{min}+} \cup U$;
14. **return** $X_C^{\mathsf{min}+}$;

Procedure chkComb

Input: Sets $\mathcal{T}$ and $X_C^{\mathsf{min}+}$ of objects.
Output: Set U of objects that are deducible from $X_C^{\mathsf{min}+}$ by γ_5.

1. $U := \emptyset$; $u_i := \emptyset$ for each $X_i \to (Y_i, N_i)$ in $\mathcal{A}$;
2. **for each** $X_i \to (Y_i, N_i)$ in $\mathcal{A}$ **do**
3. $\quad$ **for each** $o \in \mathcal{T} \cup X_C^{\mathsf{min}+}$ **do**
4. $\qquad$ **if** $o.X \subseteq X_i \cup Y_i$ **then** add o to u_i;
5. $\quad$ **if** $X_i \subseteq \bigcup_{o \in u_i} o.X$ **do**
6. $\qquad$ instantiate o_i for deducing $\bigcup_{o \in u_i} o.X$ from X_C via γ_5;
7. $\quad$ **if** $o_i.X \not\subseteq o'.X$ for all $o' \in X_C^{\mathsf{min}+}$ **do**
8. $\qquad$ add o_i to U; $X_C^{\mathsf{min}+} := X_C^{\mathsf{min}+} \setminus u_i$;
9. **return** U;

Figure 4: Algorithm QPlan

be *deduced from* another set S_1 if there exists a proof from $\bigcup_{o \in S_1} o.X$ to $\bigcup_{o \in S_2} o.X$. (b) We use γ_1–γ_5 to denote the five rules in $\mathcal{I}_E$ (Fig. 2), respectively. For instance, γ_5 denotes Combination, and $\gamma_2(X \to (Y, N))$ indicates the application of Actualization with access constraint $X \to (Y, N)$ in $\mathcal{A}$.

Algorithm QPlan also uses the following structures: (a) a set $\mathcal{B}$ of objects that are in $X_C^{\mathsf{min}+}$ but remain to be checked for other objects that can deduced from them, similar to its counterpart used in BCheck (Fig. 3); and (b) a set $\mathcal{T}$ of candidate objects deduced from equality atoms in Σ_Q, which is to be used when Combination rule is applied.

Using these structures, algorithm QPlan works as follows. It first collects in Γ all actualized constraints of $\mathcal{A}$ in the same way as BCheck (Fig. 3), and initializes both $X_C^{\mathsf{min}+}$ and $\mathcal{B}$ with the set consisting of only one object that represents the proof for X_C; it sets $\mathcal{T}$ empty (lines 1-2).

After these, QPlan iteratively finds objects that can possibly be deduced from $X_C^{\mathsf{min}+}$, by processing objects in $\mathcal{B}$ one by one (lines 3-13). For each object o in $\mathcal{B}$, it finds all possible *direct* deductions with the actualized constraints, and adds them to $\mathcal{T}$ (lines 5-7). More specifically, if there exists an actualized constraint ϕ: $W \mapsto_{\mathcal{I}_E} (Y, N)$ in Γ and if W is a subset of $o.X$, then $o.X \cup Y$ can possibly be deduced from X_C by first deducing $o.X$ using $o.\mathcal{P}$, and then deducing W by using Reflexivity (from $o.X$) followed by Transitivity (from X_C), with $o.c = N$, and possibly with Augmentation. Algorithm QPlan stores these single-step deductions in an object o_Y (line 6), and adds it to $\mathcal{T}$ for checking whether $o.X \cup Y$ is indexed in $\mathcal{A}$. It removes ϕ from Γ (line 7).

Intuitively, QPlan expands set $\mathcal{T}$ by including all new candidate objects that can *possibly* be deduced by γ_5 (*i.e.,* Combination rule), subject to the *indexing* condition of γ_5 to be checked (lines 8-11). It invokes procedure chkComb to identify combinations of objects in $\mathcal{T}$ to which γ_5 can be applied; chkComb returns a set U of new objects that encode new parameters of Q deduced by γ_5 (line 12; see details shortly). The objects of U are added to $X_C^{\mathsf{min}+}$ and $\mathcal{B}$ (line 17). The algorithm then proceeds to process the next object in $\mathcal{B}$ in the same way, until $\mathcal{B}$ becomes empty.

After the **while** loop, QPlan returns $X_C^{\mathsf{min}+}$ that contains proofs for each $X_C \mapsto_{\mathcal{I}_E} (X_Q^i, N_i)$ (line 14).

<u>*Procedure* chkComb</u>. Given $\mathcal{T}$ and $X_C^{\mathsf{min}+}$, chkComb finds all maximum subsets of $\mathcal{T} \cup X_C^{\mathsf{min}+}$ to which rule γ_5 can be applied, to deduce new parameters. More specifically, each subset satisfies the following conditions: (1) the union of their encoded attributes is indexed in $\mathcal{A}$; (2) it is *maximal*, *i.e.,* it cannot be expanded; and (3) no objects in it are already in $X_C^{\mathsf{min}+}$. Each of these subsets is encoded by a new object, representing all attributes covered by the subset.

The procedure works as follows. Assume *w.l.o.g.* that for each object o in $\mathcal{T} \cup X_C^{\mathsf{min}+}$, $o.X$ contains attributes from the same renaming S_i only. It associates a set u_i with each constraint $X_i \to (Y_i, A)$ in $\mathcal{A}$, initially empty (line 1). It collects in u_i all objects of $\mathcal{T} \cup X_C^{\mathsf{min}+}$ that can be combined using γ_5 and are indexed by $X_i \cup Y_i$ (lines 2-4). If X_i is covered by attributes encoded in the objects of u_i, then these attributes can be deduced by γ_5 and hence, a new object o_i is created to encode them (lines 5-6). If attributes in $o_i.X$ are not covered by existing objects in $X_C^{\mathsf{min}+}$, then it adds o_i to U, and removes objects of u_i from $X_C^{\mathsf{min}+}$ (lines 7-8). The process proceeds until all constraints in $\mathcal{A}$ are checked (line 2). After the loop, it returns set U.

<u>*Correctness & Complexity*.</u> The correctness of QPlan follows from Theorem 4 and the coverage property of $X_C^{\mathsf{min}+}$.

To see that QPlan is in $O(|Q|^2|\mathcal{A}|^3)$-time, observe the following. (1) At most $O(|Q||\mathcal{A}|)$ objects are added to $\mathcal{B}$. This is because each actualized constraint in Γ and each equality atom in σ_C of Q are processed *only once*; moreover, each equality atom yields at most $O(|\mathcal{A}|)$ objects. (2) The loop (lines 5-11) is executed at most $O(|Q||\mathcal{A}|)$ times *in total.* (3) Procedure chkComb is in $O(|Q||\mathcal{A}|^2)$ time; thus in the entire process, chkComb takes $O((|Q||\mathcal{A}| + |\mathcal{A}|) * |Q||\mathcal{A}|^2) = O(|Q|^2|\mathcal{A}|^3)$ time in total. Indeed, (a) its initialization is in $O(|\mathcal{A}|)$ time; (b) the total time taken by checking indexing (lines 3-4) is $O(|\mathcal{A}||\mathcal{T} \cup X_C^{\mathsf{min}+}|) = O(|Q||\mathcal{A}|^2)$; and (c) checking the conditions of line 5 and line 7 takes $O(|Q||\mathcal{A}|^2)$ time each. We remark that $|Q|$ and $|\mathcal{A}|$ are *typically small* in real-life, compared to the size of dataset D.

Example 10: We show how QPlan generates a query plan for Q_0 of Example 1 under access schema $\mathcal{A}_0$ of Example 2. Initially, both $X_C^{\mathsf{min}+}$ and $\mathcal{B}$ contain an object o_C encoding X_C such that $o_C.X = \{\mathsf{aid}, \mathsf{uid}, \mathsf{tid}_2\}$ and $o_C.\mathcal{P} = nil$. It then updates $X_C^{\mathsf{min}+}$ and $\mathcal{B}$ iteratively. At the beginning, o_C is popped off from $\mathcal{B}$. It constructs o_1 with $o_1.X = o_C.X \cup \{\mathsf{pid}_1\}$ $o_1.\mathcal{P} = [\gamma_1, \gamma_2(\mathsf{aid} \mapsto_{\mathcal{I}_E} (\mathsf{pid}_1, 1000), \gamma_4]$ and $o_1.c = 1000$; it puts o_1 in $\mathcal{T}$. Similarly, it adds o_2 to $\mathcal{T}$ with $o_2.X = o_C.X \cup \{\mathsf{fid}\}$, $o_2.\mathcal{P} = [\gamma_1,\ \gamma_2(\mathsf{uid} \mapsto_{\mathcal{I}_E} (\mathsf{fid}, 5000),$ $\gamma_3]$ and $o_2.c = 5000$. After these, it invokes chkComb and finds $U = \{o_1, o_2\}$ since $o_1.X$ and $o_2.X$ are indexed in $\mathcal{A}_0$. It replaces o_C in $\mathcal{B}$ and $X_C^{\mathsf{min}+}$ with o_1 and o_2. After that, it pops off o_1 from $\mathcal{B}$ and finds that equality atom $\mathsf{pid}_1 = \mathsf{pid}_2$ in Σ_Q is applicable to o_1. Thus it adds o_3 to $\mathcal{T}$ with $o_3.X = o_1.X \cup \{\mathsf{pid}_2\}$, $o_3.\mathcal{P} = o_1.\mathcal{P}$ and $o_3.c = 1000$. By calling chkComb, o_4 is deduced using rule γ_5, with $o_4.X = o_3.X$, $o_4.\mathcal{P} = o_3.\mathcal{P} \oplus \gamma_5$ ($\oplus$ for appending), and $o_4.c = 1000$.

Note that the parameters of in_album, friends and tagging are covered by $o_1.X$, $o_2.X$ and $o_4.X$, respectively. Hence $o_1.\mathcal{P}$, $o_2.\mathcal{P}$ and $o_4.\mathcal{P}$ tell us how to fetch subsets T_1, T_2 and T_3 from any dataset $D_0 \models \mathcal{A}_0$, 7000 tuples in total. One can verify that T_1, T_2 and T_3 are precisely those described in Example 1. As shown there, we can fetch T_1, T_2 and T_3 from D_0 and compute $Q_0(D_0)$ by using these sets only. □

5.2 Minimum D_Q

One might be tempted to search for a minimum $D_Q \subseteq D$ such that $Q(D) = Q(D_Q)$ under $\mathcal{A}$. More formally, we say that Q is *M-bounded* if for all databases D of schema R, there exists a $D_Q \subseteq D$ such that $|D_Q| \leq M$ and $Q(D) = Q(D_Q)$. It is *effectively M-bounded* if in addition, D_Q can be identified in time independent of $|D|$. These notions were referred to (efficient) scale independence in [19]. The *decision problem* for finding minimum D_Q can be stated as follows:

- Input: R, Q and $\mathcal{A}$, and a natural number M.
- Question: Is Q (effectively) M-bounded under $\mathcal{A}$?

Unfortunately, when M is part of the input, the problem for deciding (effective) boundedness becomes intractable, as opposed to quadratic-time given in Theorems 5 and 6.

Theorem 8: *It is* NP*-complete to decide whether an* SPC *query is (a) M-bounded or (b) effectively M-bounded under an access schema.* □

Proof: We sketch a proof for M-boundedness. The proof for effective M-boundedness is similar (see [5] for details).

We show the upper bound by giving an NP algorithm: guess a proof of a bounded length $(O(|Q|(|Q| + |\mathcal{A}|)))$ from Q and $\mathcal{A}$ using $\mathcal{I}_B$, and then verify the proof and check whether the bound N obtained from the proof is no larger than M; the verification and checking can be done in PTIME. We verify that it is NP-hard by reduction from the Vertex Cover problem, which is NP-complete (cf. [28]). □

6. Experimental Study

Using real-life and synthetic data, we conducted two sets of experiments to evaluate (1) the effectiveness of our query evaluation approach based on boundedness, and (2) the efficiency of algorithms BCheck, EBCheck, $\mathsf{findDP_h}$ and QPlan.

Experimental setting. We used three datasets: two real-life (TFACC and MOT) and one synthetic (TPCH).

(1) UK traffic accident (TFACC) was obtained by integrating the Road Safety Data [1], which records information about road accidents that happened in the UK from 1979 to 2005, and the National Public Transport Access Nodes data (NaPTAN) [2], with a fuzzy join on location attributes (latitude, longitude). It has 19 tables with 113 attributes, and over 89.7 million tuples in total. Its size is 21.4GB.

(2) The Ministry of Transport Test data (MOT [3]) records all MOT tests, including the makes and models of vehicles, odometer reading and reasons for failures, in year 2013. To make the data larger, we joined its 5 tables together. It is of 16.2GB size with 36 attributes and over 55 million tuples.

Synthetic data (TPCH) was generated by using TPC-H dbgen [4]. The dataset consisted of 8 relations. We varied the scale factor from 0.25 to 32 (32 by default) with the size of the data varying from 0.25GB to 32GB.

All of the three datasets were stored in MySQL.

Access schema. We manually extracted 84, 27 and 61 access constraints for access schemas of TFACC, MOT and TPCH, respectively, by examining the size of their active domains and the semantics and dependencies of their attributes. For example, on TFACC we had (1) $\mathtt{date} \to (\mathtt{aid}, 610)$ on relation R_{acc}, which states that at most 610 accidents happened in the UK during a single day from 1979 to 2005; and (2) $\mathtt{aid} \to (\mathtt{vid}, 192)$ on relation $R_{\mathtt{veh}}$, *i.e.,* at most 192 vehicles were involved in a single accident from 1979 to 2005. In fact there are many more access constraints in the datasets, which were not used in our tests. For each constraint $X \to (Y, N)$ extracted, we built index by (a) creating a table by projecting the data on attributes $X \cup Y$, and (b) building an index on X for the new table, using MySQL.

SPC *queries*. We manually designed 45 SPC queries Q on these datasets, 15 for each. The queries vary in the number #-sel of equality atoms in the selection condition σ_C of Q, which is in the range of [4, 8], and the number #-prod of Cartesian products in Q, in the range of [0, 4].

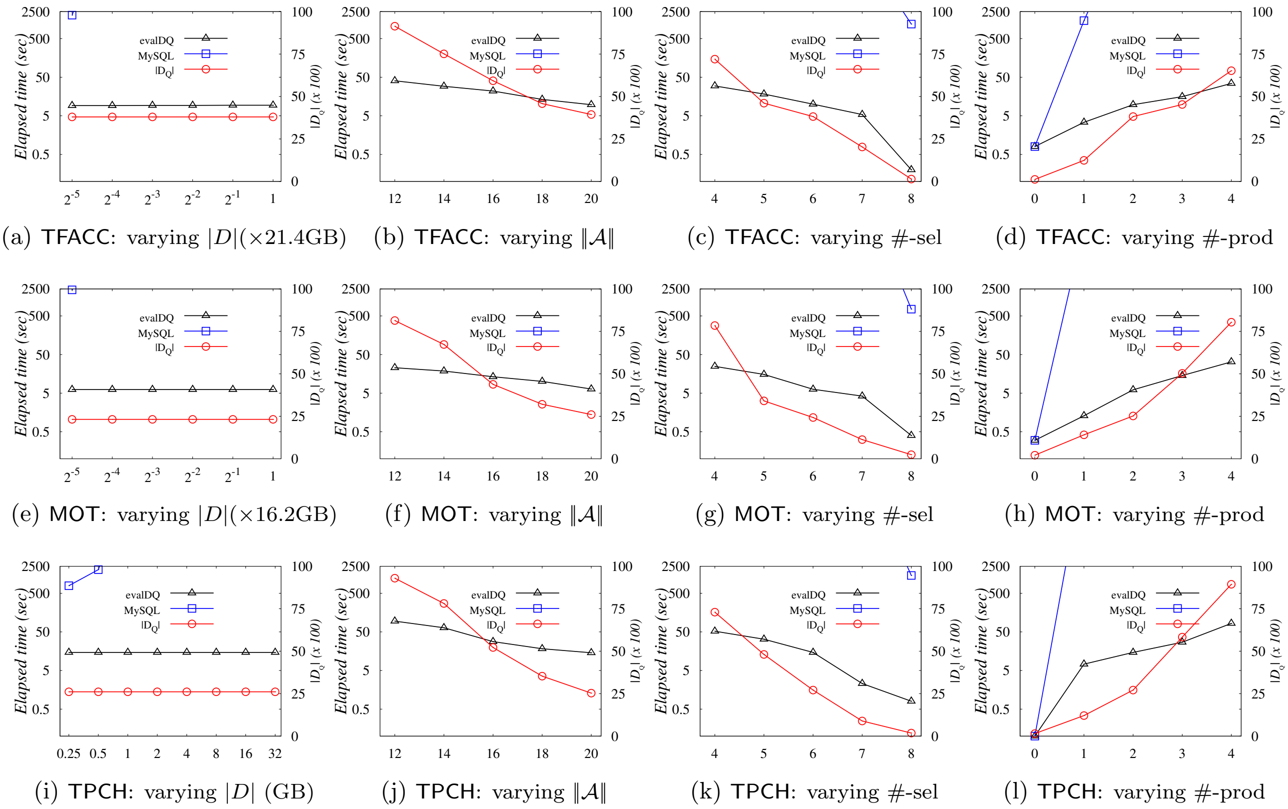
Algorithms. We implemented the following algorithms, all in Python: (1) BCheck (Section 4.1) and EBCheck (Section 4.2) for checking boundedness and effective boundedness, respectively; (2) $\mathsf{findDP_h}$ (Section 4.3) to find dominating parameters; (3) QPlan to generate query plans that identify D_Q (Section 5.1), (4) evalDQ, a simple algorithm that evaluates effectively bounded SPC queries Q following the query plans generated by QPlan, *i.e.,* fetching D_Q from D and evaluating Q on D_Q, and (5) MySQL, which directly uses MySQL for query evaluation, with all the indices specified in $\mathcal{A}$.

The experiments were conducted on an Amazon EC2 high-memory instance with 17GB memory and 6.5 EC2 compute units. We used MySQL 5.5.35 and MyISAM engine. All the experiments were run 3 times. The average is reported here.

Experimental Results. We next report our findings.

Exp-1: Effectiveness of bounded query evaluation. The first set of experiments evaluated the effectiveness of the bounded query evaluation approach. We first examined the queries generated by using algorithm EBCheck. We found that 35 out of 45 queries are effectively bounded under the access schemas, over 77%. We then evaluated the effectiveness of the query plans generated by QPlan, by comparing the running time of evalDQ with its counterpart of MySQL. The results are reported in Figures 5, on datasets TFACC, MOT and TPCH, by varying $|D|$, Q and $\|\mathcal{A}\|$ (we use $\|\mathcal{A}\|$ to denote the number of access constraints in $\mathcal{A}$). In each of them, we report (a) the average evaluation time (*the left y-axis*), and (b) the size $|D_Q|$ of datasets D_Q accessed by evalDQ (*the right y-axis*). Unless stated otherwise, the tests were conducted on all effectively bounded queries, all access constraints, and full-size datasets by default.

(1) Impact of $|D|$. To evaluate the impact of $|D|$, we varied the size of TFACC and MOT by using scale factors from 2^{-5} to 1, and varied TPCH from 0.25GB to 32GB.



(a) TFACC: varying $|D|(\times 21.4\text{GB})$ (b) TFACC: varying $\|\mathcal{A}\|$ (c) TFACC: varying #-sel (d) TFACC: varying #-prod

(e) MOT: varying $|D|(\times 16.2\text{GB})$ (f) MOT: varying $\|\mathcal{A}\|$ (g) MOT: varying #-sel (h) MOT: varying #-prod

(i) TPCH: varying $|D|$ (GB) (j) TPCH: varying $\|\mathcal{A}\|$ (k) TPCH: varying #-sel (l) TPCH: varying #-prod

Figure 5: Effectiveness of bounded query evaluation

The results are shown in Figures 5(a), 5(e) and 5(i), which tell us the following. (1) The evaluation time of evalDQ is *independent of the size of* D. This verifies our analysis in Section 5. (2) MySQL does not scale well with large D. Indeed, evalDQ consistently took 9.3s, 6.2s, 14.7s on TFACC, MOT and TPCH, respectively, no matter how large the parts of the datasets were used. In contrast, MySQL took 2024s, 2367s and 2045s on subsets of TFACC, MOT and TPCH of sizes $2^{-5} \times 21.4$GB, $2^{-5} \times 16.2$GB and 0.5GB, respectively, and *could not* finish its computation within 2500s for *all larger subsets*. For example, MySQL took longer than 14 hours on the entire TFACC. That is why only a couple of points are reported for MySQL in the figures. That is, even on *the smallest subsets* we tested, MySQL was 10^2 *times slower* than evalDQ, and at least 5.4×10^3 *time slower* on full sized dataset. In fact, *the larger the datasets* are used, *the bigger the gap* between MySQL and evalDQ are. (3) The size $|D_Q|$ of data accessed evalDQ is also *independent on* $|D|$. Indeed, evalDQ accessed 3800, 2320, 2610 tuples on average, on TFACC, MOT and TPCH, respectively, on all subsets.

(2) Impact of $\|\mathcal{A}\|$. To evaluate the impact of access constraints, we varied $\|\mathcal{A}\|$ from 12 to 20 and tested the queries that are effectively bounded. Accordingly we varied the indices used by MySQL. The results are shown in Figures 5(b), 5(f) and 5(j). The results tell us the following. (1) More access constraints help QPlan get better query plans. For example, when 20 access constraints were used, evalDQ took 9.6s, 6.4s and 14.4s for queries on TFACC, MOT and TPCH, respectively, as opposed to 40.4s, 22.8s and 95s with 12 access constraints, although queries are effectively bounded in both cases. (2) The more access constraints are used, the smaller $|D_Q|$ is, as QPlan can find better proofs (query plans) given more options. (3) MySQL did not produce results *in any single test* within 2500s, no matter whether we used more or less indices embedded in access schemas.

Algorithm	TFACC	MOT	TPCH
BCheck	0.8s	0.3s	0.5s
EBCheck	0.8s	0.3s	0.5s
findDP_h	0.3s	0.1s	0.2s
QPlan	2.1s	0.9s	1.4s

Table 1: Elapsed Time

(3) Impact of Q. To evaluate the impact of queries, we varied #-sel of Q from 4 to 8, and #-prod of Q from 0 to 4. We report the average evaluation time of evalDQ and the size $|D_Q|$ for all queries with the same #-sel or #-prod, in Figures 5(c), 5(g) and 5(k), and Figures 5(d), 5(h) and 5(l), respectively. They tell us the following. (1) The complexity of Q has impacts on the quality of query plans generated by QPlan. The larger #-sel or the smaller #-prod is, the better the evaluation time of evalDQ and the size $|D_Q|$ of data accessed by evalDQ, as expected. (2) Algorithm evalDQ scales well with #-sel and #-prod. It finds answers in all cases within 90s, on the three full datasets. (3) MySQL is indifferent to #-sel. But it is sensitive to #-prod: it is as fast as evalDQ when #-prod $= 0$, *i.e.,* when there is *no* Cartesian product at all; but it cannot stop within 2500s for queries even with 1 Cartesian product, except one case of TFACC.

Exp-2: Efficiency. The second set of experiments evaluated the efficiency of our algorithms BCheck, EBCheck, findDP_h and QPlan on queries and access schemas for each of TFACC, MOT and TPCH. We used all access constraints, and report in Table 1 *the longest elapsed time* of each algo-

Problem	M is not predefined	M is part of input
$\mathsf{Bnd}(Q, \mathcal{A})$	$O(\lvert Q\rvert(\lvert\mathcal{A}\rvert + \lvert Q\rvert))$ (Th 5)	NP-complete (Th 8)
$\mathsf{EBnd}(Q, \mathcal{A})$	$O(\lvert Q\rvert(\lvert\mathcal{A}\rvert + \lvert Q\rvert))$ (Th 6)	NP-complete (Th 8)
$\mathsf{DP}(Q, \mathcal{A})$	NP-complete (Th 7)	NP-complete [5]
$\mathsf{MDP}(Q, \mathcal{A})$	NPO-complete (Th 7)	NPO-complete [5]

Table 2: Complexity bounds

rithm on all queries for each dataset. These results verify that all of our algorithms are efficient: for all queries, all of our algorithms took no more than 2.1 seconds, even QPlan, the one with the highest complexity (see Section 5). These confirm our complexity analyses of these algorithms.

Summary. From the experimental results we find the following. (1) The notion of effective boundedness is practical. It is rather easy to find sufficiently many access constraints in real-life data, and many practical queries are actually effectively bounded. (2) The bounded query evaluation approach allows us to query big data. Its evaluation time and amount of data accessed are *independent of* the size of the underlying dataset. For example, on a real-life dataset of 21.4GB, evalDQ finds answers to queries in 9.3 seconds by accessing no more than 3800 tuples on average. In contrast, MySQL is unable to get answers within 2500 seconds in almost all of the cases except for extremely restricted queries (without Cartesian products). Even on a dataset of $2^{-4} \times 21.4$GB (1.3GB), it took longer than 3 hours. The gap between evalDQ and MySQL is *more substantial* on larger datasets. (3) Our algorithms are efficient: they are able to check (effective) boundedness, identify dominating parameters, and generate query plans in 2.1 seconds for queries defined on large schemas and a variety of access constraints.

7. Conclusion

We have studied (effective) boundedness for SPC, a class of queries commonly used in practice. We have investigated fundamental problems to characterize what SPC query Q can be evaluated under an access schema $\mathcal{A}$ by accessing a bounded dataset, and to make Q effectively bounded under $\mathcal{A}$ by identifying a minimum set of parameters to instantiate. We have established their complexity bounds, as summarized in Table 2. We have also developed efficient (heuristic) algorithms to make practical use of effective boundedness. Our experimental results have verified that effective boundedness yields a promising approach to querying big data.

Several extensions are targeted for future work. (1) It is undecidable to determine whether an $\mathcal{RA}$ (relational algebra) query is (effectively) bounded [19], and hence, it is impossible to characterize (effectively) bounded $\mathcal{RA}$ queries. Nonetheless, we can still find efficient heuristic algorithms to check whether $\mathcal{RA}$ queries are effectively bounded. (2) When it is cost prohibitive to compute exact answers, we need to develop efficient algorithms to compute *approximate* answers by accessing a bounded dataset. (3) Given a set of parameterized queries, we want to study how to build an *optimal* access schema under which the queries are effectively bounded. (4) When a query is not effectively bounded, it may be effectively bounded *incrementally* or *using views*. A preliminary study of these issues has been reported in [11,19]. However, effective algorithms remain to be developed.

8. References

[1] *http://data.gov.uk/dataset/road-accidents-safety-data.*
[2] *http://data.gov.uk/dataset/naptan.*
[3] *http://data.gov.uk/dataset/anonymised_mot_test.*
[4] *http://www.tpc.org/tpch/.*
[5] Full version. *http://homepages.inf.ed.ac.uk/s1165433/bounded.pdf.*
[6] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
[7] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, 1999.
[8] 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.
[9] 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.
[10] 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.
[11] M. Armbrust, E. Liang, T. Kraska, A. Fox, M. J. Franklin, and D. Patterson. Generalized scale independence through incremental precomputation. In *SIGMOD*, 2013.
[12] G. Ausiello. *Complexity and approximation: Combinatorial optimization problems and their approximability properties.* Springer, 1999.
[13] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, 2003.
[14] V. Bárány, M. Benedikt, and P. Bourhis. Access patterns and integrity constraints revisited. In *ICDT*, 2013.
[15] A. Deutsch, B. Ludäscher, and A. Nash. Rewriting queries using views with access patterns under integrity constraints. *TCS*, 371(3), 2007.
[16] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996.
[17] Facebook. *http://newsroom.fb.com.*
[18] Facebook. Constraints on the number of photos, friends and tags. *https://www.facebook.com/help {/227794810567981, /community/question/?id=364005057055660}*, 2014.
[19] W. Fan, F. Geerts, and L. Libkin. On scale independence for querying big data. In *PODS*, 2014.
[20] W. Fan, F. Geerts, and F. Neven. Making queries tractable on big data with preprocessing. In *VLDB*, 2013.
[21] M. N. Garofalakis and P. B. Gibbons. Wavelet synopses with error guarantees. In *SIGMOD*, 2004.
[22] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In *SODA*, 1999.
[23] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *VLDB*, 1999.
[24] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, 2009.
[25] P. C. Kanellakis. On the computational complexity of cardinality constraints in relational databases. *Inf. Process. Lett.*, 11(2):98–101, 1980.
[26] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley, 1990.
[27] C. Li. Computing complete answers to queries in the presence of limited access patterns. *VLDB J.*, 12(3), 2003.
[28] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.
[29] P. Rösch and W. Lehner. Sample synopses for approximate answering of group-by queries. In *EDBT*, 2009.
[30] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD*, 1999.

Appendix A: Proofs of Characterizations of Boundedness

Proof of Theorem 3

To verify Theorem 3, we first introduce the notion of *access closures*, based on which we then give two lemmas. From the lemmas, Theorem 3 naturally follows.

Definition 1: Given an access schema $\mathcal{A}$ and an SPC query Q defined on a set U of attributes, the *access closure* of a set X of attributes under $\mathcal{A}$ for Q, denoted by X^*, is the set consisting of all attributes y in U such that for all $D \models \mathcal{A}$, there exists $D' \subseteq D$ such that

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

(b) for all X-values $\bar{a}$ in D, $|\pi_y \sigma_{X=\bar{a}}(D)| \le N_y$ for some positive integer N_y determined by $\mathcal{A}$ and Q only.

Here $\sigma_{X=\bar{a}}(D)$ is short for $\sigma_{X_1=\bar{a}_1 \wedge \cdots \wedge X_n=\bar{a}_n}(S_1 \times \cdots \times S_n)(D)$, where for each $i \in [1, n]$, (i) $X_i \subseteq X$ with attributes all from S_i; and (ii) $\bar{a}_i$ consists of values of $\bar{a}$ that correspond to X_i. We will also use $\pi_Y(D)$ to represent $\pi_Y(S_1 \times \cdots \times S_n)(D)$. $\square$

It suffices to show the following. Consider an SPC query $Q(Z)$, where Z is the set of parameters (see Section 2).

Lemma 9: *An* SPC *query $Q(Z)$ is bounded under access schema $\mathcal{A}$ if and only if $Z \cup X_B \subseteq (X_B \cup X_C)^*$.* $\square$

Lemma 10: *For any sets X and Y of attributes of Q, $X \mapsto_{\mathcal{I}_B} (Y, N)$ for some integer N if and only if $Y \subseteq X^*$.* $\square$

For if these lemmas hold, $Q(Z)$ is bounded under $\mathcal{A}$ if and only if $Z \cup X_B \subseteq (X_B \cup X_C)^*$ if and only if $X_B \cup X_C \mapsto_{\mathcal{I}_B} (Z \cup X_B, N)$ for some N, *i.e.*, $X_B \cup X_C$ covers $Q(Z)$. From these Theorem 3 follows.

Below we first prove Lemma 9, followed by the proof of Lemma 10.

Proof of Lemma 9.

$\boxed{\Rightarrow}$ Assume that $Q(Z)$ is bounded. Then by the definition of boundedness, for any $D \models \mathcal{A}$, there exists $D' \subseteq D$ such that $|D'| \le M$ for some positive integer M and moreover, $Q(D) = Q(D')$. Therefore, $|\pi_z \sigma_{X_B \cup X_C=\bar{a}}(S_1 \times \cdots \times S_n)(D')| \le M$, for any parameter $z \in Z$ in Q and any $(X_B \cup X_C)$-value $\bar{a}$. By the definition of $(X_B \cup X_C)^*$, we have that $z \in (X_B \cup X_C)^*$. Moreover, for any $x \in X_B$, $x \in (X_B \cup X_C)^*$. Therefore, $Z \cup X_B \subseteq (X_B \cup X_C)^*$.

$\boxed{\Leftarrow}$ Conversely, suppose that $Z \cup X_B \subseteq (X_B \cup X_C)^*$. Thus, $Z \subseteq (X_B \cup X_C)^*$. By the definition of access closures, for each z_i in $Z = \{z_1, \ldots, z_m\}$, and for any $D \models \mathcal{A}$, there exists $D'_{z_i} \subseteq D$ such that $Q(D'_{z_i}) = Q(D)$ and $|\pi_{z_i} \sigma_{X_B \cup X_C=\bar{a}}(S_1 \times \cdots \times S_n)(D'_{z_i})| \le N_{z_i}$ for some integer N_{z_i} and any $(X_B \cup X_C)$-value $\bar{a}$.

We first show that there exists $D_Q \subseteq D$ such that $Q(D_Q) = Q(D)$ and for each z_i in Z, $|\pi_{z_i} \sigma_{X_B \cup X_C=\bar{a}}(S_1 \times \cdots \times S_n)(D_Q)| \le N_{z_i}$. We show this by induction on the number $|Z|$ of parameters in Z. For the induction basis, by the definition of access closures, we have that for any $D \models \mathcal{A}$, there exists $D_{z_1} \subseteq D$ such that $Q(D_{z_1}) = Q(D)$ and $|\pi_{z_1} \sigma_{X_B \cup X_C=\bar{a}}(S_1 \times \cdots \times S_n)(D_{z_1})| \le N_{z_1}$. Suppose that for any $i \le k$, there exists $D_{z_i} \subseteq D$ such that $Q(D_{z_i}) = Q(D)$ and for any $i \in [1, k]$, $|\pi_{z_i} \sigma_{X_B \cup X_C=\bar{a}}(S_1 \times \cdots \times S_n)(D_{z_i})| \le N_{z_i}$. We next show that there exists a subset $D_{z_{k+1}} \subseteq D$ such that $Q(D_{z_{k+1}}) = Q(D)$ and $|\pi_{z_i} \sigma_{X_B \cup X_C=\bar{a}}(S_1 \times \cdots \times S_n)(D_{k+1})| \le N_{z_i}$, for any $i \in [1, k+1]$. Indeed, by $D_{z_k} \subseteq D$, $D_{z_k} \models \mathcal{A}$. Thus by the definition of access closures, there exists a subset $D_{z_{k+1}} \subseteq D_{z_k}$ such that $Q(D_{z_{k+1}}) = Q(D_{z_k}) = Q(D)$ and $|\pi_{z_{k+1}} \sigma_{X_B \cup X_C=\bar{a}}(S_1 \times \cdots \times S_n)(D_{z_{k+1}})| \le N_{z_{k+1}}$, by taking D_{z_k} as D. By the induction hypothesis and the monotonicity of SPC queries, the statement holds for $k+1$. Therefore, there must exist $D_Q \subseteq D$ such that $Q(D_Q) = Q(D)$ and for any z_i in $Z = \{z_1, \ldots, z_m\}$, $|\pi_{z_i} \sigma_{X_B \cup X_C=\bar{a}}(S_1 \times \cdots \times S_n)(D_Q)| \le N_{z_i}$. As a result, $|\pi_Z \sigma_{X_B \cup X_C=\bar{a}}(S_1 \times \cdots \times S_n)(D_Q)| \le \prod_{i=1}^{m} N_{z_i}$, and $Q(D_Q) = Q(D)$.

Based on the analysis we can see the following.

(1) For any $D \models \mathcal{A}$, if $Q(D) \neq \emptyset$, then there exist some X_B-value $\bar{x}$ and constants $\bar{c}$ such that $Q(D) = \pi_Z \sigma_{X_B=\bar{x} \wedge X_C=\bar{c}}(S_1 \times \cdots \times S_n)(D)$. By the induction above, there exists $D'_x \subseteq \sigma_{X_B=\bar{x} \wedge X_C=\bar{c}}(S_1 \times \cdots \times S_n)(D)$, such that $|\pi_Z(D'_x)| \le \prod_{i=1}^{m} N_{z_i}$ and $Q(D'_x) = Q(\sigma_{X_B=\bar{x} \wedge X_C=\bar{c}}(D)) = Q(D)$. Define D_Q^x as follows: (a) D_Q^x consists of at most $|\pi_Z \sigma_{X_B=\bar{x} \wedge X_C=\bar{c}}(D'_x)| \le \prod_{i=1}^{m} N_i$ tuples; (b) $\pi_{X_C}(D_Q^x) = \bar{c}$ and $\pi_{X_B}(D_Q^x) = \bar{x}$; and (c) $\pi_Z(D_Q^x) = \pi_Z \sigma_{X_B=\bar{x} \wedge X_C=\bar{c}}(D'_x)$. It is easy to see that such D_Q^x exists. Then $Q(D_Q^x) = Q(D'_x) = Q(D)$ and $|D_Q^x| \le \prod_{i=1}^{m} N_i$. Moreover, query answer $Q(D_Q^x)$ is not dependent on the choice of $\bar{x}$ since for any x_B in X_B and any z in Z, $\Sigma_Q \not\models x = z$. Hence, as remarked in Section 3, X_B only involves in Boolean condition checking and $\bar{x}$ just serves as a witness for the truth value of the condition. That is, $Q(D_Q^x)$ is determined by X_C value $\bar{c}$, which is given by Q. Thus, whenever $Q(D) \neq \emptyset$, there must exist $\bar{x} = \bar{x}_B$ such that D_Q is determined by Q only such that $Q(D_Q) = D(D_Q^{\bar{x}_B})$ and $|D_Q| = |D_Q^{\bar{x}_B}| \le \prod_{i=1}^{m} N_{z_i}$.

(2) If $Q(D) = \emptyset$, then let $D_Q = \emptyset$. We have $Q(D_Q) = Q(\emptyset) = \emptyset = Q(D)$.

Putting these together, for any $D \models \mathcal{A}$, there exists $D_Q \subseteq D$ such that $|D_Q| \le \prod_{i=1}^{m} N_i$ and $Q(D_Q) = Q(D)$, where N_i is determined by Q and $\mathcal{A}$ only. Hence $Q(Z)$ is bounded under $\mathcal{A}$. $\square$

Algorithm compAC

Input: Actualized access constraints Γ, an SPC query Q, and a set X of attributes in Q.
Output: The access closure X^* of X under Γ for Q.

1. **for each** $x = y$ in Σ_Q **do**
   /* Σ_Q is the set of all equality items in the selection condition σ_C of Q */
2.   Rename all y appearing in Γ and Σ_Q to x; /*according to some order*/
3. *unused* $:= \Gamma$; *closure* $:= X$;
4. **repeat until** no further changes
5.   **if** $W \to (V, N) \in$ *unused* and $W \subseteq$ *closure* **then**
6.   *unused* $:=$ *unused* $\setminus \{W \to (V, N)\}$;
7.   *closure* $:=$ *closure* $\cup V$;
8. **for each** pair (x, y) such that y has been renamed to x **do**
9.   **if** $x \in$ *closure* **do**
10.   *closure* $:=$ *closure* $\cup \{y\}$;
11. **return** *closure*;

Figure 6: Algorithm compAC for computing access closure

Proof of Lemma 10.

We now show that $X \mapsto_{\mathcal{I}_B} (Y, N)$ if and only if $Y \subseteq X^*$.

$\boxed{\Rightarrow}$ We first show that if $X \mapsto_{\mathcal{I}_B} (Y, N)$ for some natural number N, then $Y \subseteq X^*$. We prove this by induction on the length of *proofs* $s = s_1 \circ s_2 \circ \cdots \circ s_n$ for $X \mapsto_{\mathcal{I}_B} (Y, N)$ from $\mathcal{A}$ and Q; here to simplify the discussion, each s_i denotes the application of one of the rules in $\mathcal{I}_B$ (Fig. 1).

(1) *Basis:* when $n = 1$, $X \mapsto_{\mathcal{I}_B} (Y, N)$ is derived by a single step $s = s_1$. Obviously, s_1 can only be the *reflexivity rule* or the *actualization rule*. By the definitions of the rules and access closures, $Y \subseteq X^*$.

(2) *Inductive step:* assume that if $X_k \mapsto_{\mathcal{I}_B} (Y_k, N_k)$ can be derived in $n = k$ steps, *i.e.,* via a proof $s_1 \circ s_2 \circ \cdots \circ s_k$, then $Y_K \subseteq X_K^*$. Consider that $X_{k+1} \mapsto_{\mathcal{I}_B} (Y_{k+1}, N_{k+1})$ is derived in $k+1$ steps, via proof $s_1 \circ \cdots \circ s_k \circ s_{k+1}$. We show that $Y_{k+1} \subseteq X_{k+1}^*$, by considering the following cases.

(i) If s_{k+1} is the reflexivity rule or actualization rule in $\mathcal{I}_B$, then $X_{k+1} \mapsto_{\mathcal{I}_B} (Y_{k+1}, N_{k+1})$ can be derived in 1 step and thus $Y_{k+1} \subseteq X_{k+1}^*$.

(ii) If s_{k+1} is the augmentation rule, then there exists $X' \mapsto_{\mathcal{I}_B} (Y', N')$ that is derived within k steps via $s_1 \circ \cdots \circ s_k$, and there exists a set W of attributes such that $X'W = X_{k+1}$ and $YW = Y_{k+1}$. By the inductive hypothesis, we have that $Y' \subseteq X'^*$. That is, for any $y \in Y'$ and any $D \models \mathcal{A}$, there exists $D' \subseteq D$ such that $|D'_y(X' = \bar{a})| \leq N_y$ for some integer N_y, and moreover, $Q(D) = Q(D')$. Thus, for any $y' \in Y' \cup W$, $|D'_{y'}(X' = \bar{a} \wedge W = \bar{b})| \leq N_y$, for any $\bar{b}$ for W in D'. Thus, $Y_{k+1} \subseteq X_{k+1}^*$. Similarly, one can verify the statement when s_{k+1} is the transitivity rule.

(iii) If s_{k+1} is the X-equality rule. Then there exists $X_k \mapsto_{\mathcal{I}_B} (Y_k, N_k)$ that can be derived in k steps via $s_1 \circ \cdots \circ s_k$, and there exists $X'' \subseteq X_k$ such that $X' = X''$, $X_{k+1} = (X_k \setminus X'') \cup X'$, $Y_{k+1} = Y_k$. By the inductive hypothesis, $Y_{k+1} = Y_k \subseteq X_K^*$. That is, for any $y \in Y_{k+1}$ and any $D \models \mathcal{A}$, there exists $D' \subseteq D$ such that $|D'_y(X_K = \bar{a})| \leq N_y$ for some integer N_y, and moreover, $Q(D) = Q(D')$. Since $(X_k \setminus X'') \cup X' = X_{k+1}$, thus $|D'_y(X_{k+1} = \bar{a})| \leq N_y$ for some integer N_y, for any $y \in Y_{k+1}$. That is $Y_{k+1} \subseteq X_{k+1}^*$.

(iv) If s_{k+1} is the Y-equality rule, then there exists $X_k \mapsto_{\mathcal{I}_B} (Y_k, N_k)$ that can be deduced in k steps via $s_1 \circ \cdots \circ s_k$, and there exists $Y'' \subseteq Y_k$ such that $Y' = Y''$, $X_{k+1} = X_k$, and $Y_{k+1} = Y_k \cup Y'$. By the inductive hypothesis, $Y_k \subseteq X_k^*$. That is, for any $y \in Y_k$ and any $D \models A$, there exists $D' \subseteq D$ such that $|D'_y(X_{k+1} = X_k = \bar{a})| \leq N_y$ for some integer N_y, and moreover, $Q(D) = Q(D')$. For any $y \in Y_{k+1}$, if $y \in Y_k$, then $|D'_y(X_{k+1} = \bar{a})| \leq N_y$; if $y \in Y'$, there must exist $y_k \in Y_k$ such that $D'_y(X = \bar{a}) = D'_{y_k}(X = \bar{a})$, *i.e.,* $|D'_y(X = \bar{a})| \leq N_{y_k}$. Thus $Y_{k+1} \in X_k^* = X_{k+1}^*$.

Therefore, the statement holds for $k+1$.

$\boxed{\Leftarrow}$ We next show that if $Y \subseteq X^*$, then $X \mapsto_{\mathcal{I}_B} (Y, N)$ for some positive integer N. We first show that, for any set X of attributes of Q, there exists a proof s for $X \mapsto_{\mathcal{I}_B} (X^*, N)$. Since $Y \subseteq X^*$, we then have a proof $s \circ \varphi \circ \varphi'$ that entails $X \mapsto_{\mathcal{I}_B} (Y, N)$, where φ and φ' are the reflexivity rule and the transitivity rule, respectively.

To show that $X \mapsto_{\mathcal{I}_B} (X^*, N)$ for some N, we first provide algorithm compAC shown in Fig. 6 that, given any access schema $\mathcal{A}$, any SPC query Q and any set X of variables in Q, computes X^* under $\mathcal{A}$ for Q. One can verify that algorithm compAC correctly computes X^*.

We then show that $X \mapsto_{\mathcal{I}_B} (X^*, N)$ for some N. Let *closure*$_i$ be the value of variable *closure* after i iterations of step 4 (line 4 of Algorithm compAC) for some execution on input $\mathcal{A}$, Q and X. We prove that $X \mapsto_{\mathcal{I}_B} (X^*, N)$ by induction on i. Initially, set *closure*$_0 = X$.

(1) Basis: $X \mapsto_{\mathcal{I}_B} (closure_0, N_0)$ follows from the *reflexivity rule*.

(2) Inductive step: suppose that a *proof* $s_{k_i} = s_1 \circ \cdots \circ s_{k_i}$ has been constructed for $X \mapsto_{\mathcal{I}_B} (closure_i, N_i)$ for some N_i. Suppose further that $W \to (V, N)$ is chosen for the $(i+1)^{st}$ iteration. It follows that $W \subseteq closure_i$ and $closure_{i+1} = closure_i \cup V$. Consider the following four cases.

Case (1): All variables in W and V are not rewritten. In this case, extend the proof s_i by adding the following steps:

- let σ_{k_i+1} be the actualization rule, leading to $W \mapsto_{\mathcal{I}_B} (V, N_1)$ for some N_1;
- let σ_{k_i+2} be the reflexivity rule, leading to $closure_i \mapsto_{\mathcal{I}_B} (W, N_2)$ for some N_2;
- let σ_{k_i+3} be the transitivity rule, leading to $closure_i \mapsto_{\mathcal{I}_B} (V, N_3)$ for some N_3;
- let σ_{k_i+4} be the augmentation rule, leading to $closure_i \mapsto_{\mathcal{I}_B} (closure_{i+1}, N_4)$ for some N_4;
- let σ_{k_i+5} be the transitively rule, leading to $X \mapsto_{\mathcal{I}_B} (closure_{i+1}, N_i * N_4)$.

Case (2): A subset W_1 of W has been written from W_2 due to the equality items $W_1 = W_2$, *i.e.,* $(W \setminus W_1) \cup W_2 \to (V, N)$ is in $\mathcal{A}$ before variable rewriting. In this case, extend the proof s_i by adding the following steps:

- let σ_{k_i+1} be the actualization rule, leading to $(W \setminus W_1) \cup W_2 \mapsto_{\mathcal{I}_B} (V, N_1)$ for some N_1;
- let σ_{k_i+2} be the X-equality rule, leading to $W \mapsto_{\mathcal{I}_B} (V, N_2)$ for some N_2;
- let σ_{k_i+3} be the reflexivity rule, leading to $closure_i \mapsto_{\mathcal{I}_B} (W, N_3)$ for some N_3;
- let σ_{k_i+4} be the transitivity rule, leading to $closure_i \mapsto_{\mathcal{I}_B} (V, N_4)$ for some N_4;
- let σ_{k_i+5} be the augmentation rule, leading to $closure_i \mapsto_{\mathcal{I}_B} (closure_{i+1}, N_5)$ for some N_5;
- let σ_{k_i+6} be the transitively rule, leading to $X \mapsto_{\mathcal{I}_B} (closure_{i+1}, N_i * N_5)$.

Case (3): A subset V_1 of V has been written from V_2 due to the equality selection condition $V_1 = V_2$ in Q, *i.e.,* $W \to (V', N)$ is in $\mathcal{A}$ before variable rewriting, where $V = V' \cup V_2$ and $V_1 \subseteq V'$. In this case, extend the proof s_i by adding the following steps:

- let σ_{k_i+1} be the actualization rule, leading to $W \mapsto_{\mathcal{I}_B} (V', N_1)$ for some N_1;
- let σ_{k_i+2} be the Y-equality rule, leading to $W \mapsto_{\mathcal{I}_B} (V, N_2)$ for some N_2;
- let σ_{k_i+3} be the reflexivity rule, leading to $closure_i \mapsto_{\mathcal{I}_B} (W, N_3)$ for some N_3;
- let σ_{k_i+4} be the transitivity rule, leading to $closure_i \mapsto_{\mathcal{I}_B} (V, N_4)$ for some N_4;
- let σ_{k_i+5} be the augmentation rule, leading to $closure_i \mapsto_{\mathcal{I}_B} (closure_{i+1}, N_5)$ for some N_5;
- let σ_{k_i+6} be the transitively rule, leading to $X \mapsto_{\mathcal{I}_B} (closure_{i+1}, N_i * N_5)$.

Case (4): Both variables in W and V have been rewritten. The extension is a combination of extensions of cases (2) and (3).

Thus, $X \mapsto_{\mathcal{I}_B} (X^*, N)$. This completes the proof of Lemma 10 and hence, the proof of Theorem 3. □

Proof of Theorem 4

To prove Theorem 4, we need a formalization of effectively bounded queries by defining the allowed operations for accessing data via indices under access schema, and use the fetched data for query answering. More specifically, we require that a query Q is effectively bounded under $\mathcal{A}$ only if

(1) for each attribute occurrence A in X_Q^i of R_i of Q, there exists a nonempty sequence of fetching operations $(\mathsf{fetch}_{X \cup Y}(X \to (Y, N), \bar{x})$ and checking-combination operations $(\mathsf{chkComb}_Z(X \to (Y, N), S_{\bar{Z}})$ of bounded length under access constraints in $\mathcal{A}$ such that (a) $\bar{x}$ is a X-value for fetch and $S_{\bar{Z}}$ is a set of Z-values that are indexed by $X \to (Y, N)$, and (b) (with projection) we get a set S_A of A-values for A;

(2) For each X_Q^i of Q, let D_Q^i consist of values returned by $\mathsf{chkComb}(\phi, \bar{c} \in \times_{A \in X_Q^i} S_A)$, for some access constraint ϕ in $\mathcal{A}$ that indexes X_Q^i; and

(3) $Q(D) = \pi_Y \sigma_C (D_Q^1 \times \cdots \times D_Q^n)$.

That is, we assume that, when Q is effectively bounded under $\mathcal{A}$, $Q(D)$ can be fetched and computed from D using indices under access constraints in $\mathcal{A}$, in the above simple way.

Based on the formal specification of effective boundedness, we then introduce the notion of *effective access closures* X^+ for a set X of attributes, which is a revision of access closures used above. Finally, we give two lemmas based on the notion, from which Theorem 4 follows.

Definition 2: Given an access schema $\mathcal{A}$ and an SPC query $Q(Z) = \pi_Z \sigma_C (S_1 \times \ldots \times S_n)$ defined on a set U of attributes, the *effective access closure* of a set X of attributes under $\mathcal{A}$ for $Q(Z)$, denoted by X^+, is the set Y of all subsets of attributes in S_i for all $i \in [1, n]$, such that for any database $D \models \mathcal{A}$, there exists $D' \subseteq D$ that satisfies the following conditions:

- $Q(D) = Q(D')$; and
- for any X-value $\bar{a}$ ***in*** D, $|\pi_Y \sigma_{X=\bar{a}} (S_1 \times \cdots \times S_n)(D')| \leq N_Y$ for some positive integer N_Y determined by $\mathcal{A}$ and Q only,

independent of $|D|$; and

- $\circ$ D' can be identified within time T_Y that is determined by $\mathcal{A}$ and Q, independent of $|D|$, under the above specification of effective boundedness. $\square$

It suffices to show the following two lemmas.

Lemma 11: *An* SPC *query* $Q(Z) = \pi_Z \sigma_C(S_1 \times \cdots \times S_n)$ *is effectively bounded under an access schema* $\mathcal{A}$ *if and only if for each* $i \in [1, n]$,

(a) X_Q^i *is indexed under* $\mathcal{A}$*; and*

(b) $X_Q^i \in X_C^+$.

Here X_Q^i *is the set of all attributes of* S_i *that appear in either* σ_C *or* Z *of* Q. $\square$

Lemma 12: *For any access schema* $\mathcal{A}$*, any* SPC *query* Q *and any sets* X *and* Y *of attributes such that* Y *is a subset of attributes of some relation* S_i *in* Q*,* $X \mapsto_{\mathcal{I}_E} (Y, N)$ *for some integer* N *if and only if* $Y \in X^+$. $\square$

For if there hold, then $Q(Z)$ is effectively bounded iff X_C^i is indexed and $X_Q^i \in X_C^+$ iff X_C^i is indexed and $X \mapsto_{\mathcal{I}_E} (Y, N)$ for some number N, *i.e.,* Theorem 4 holds.

We now prove Lemmas 11 and 12 one by one.

Proof for Lemma 11.

$\boxed{\Rightarrow}$ Assume that $Q(Z)$ is effectively bounded. By the definition of effective boundedness, for any database $D \models \mathcal{A}$, one can compute a $D_Q \subseteq D$ in time independent of $|D|$ such that $Q(D_Q) = Q(D)$ and $|D_Q| \leq M$ for a bound M independent of $|D|$. Let $D' = D_Q$. Then one can find the following: (1) $Q(D') = Q(D_Q) = Q(D)$; (2) for any X_C-value $\bar{a}$, $\pi_{X_Q^i} \sigma_{X_C = \bar{a}}(S_1 \times \cdots \times S_n)(D')$ consists of no more than M distinct X_Q^i values, for each S_i in Q; and (3) $D' = D_Q$ can be identified in time independent of $|D|$ given the assumption. Thus, by the definition of effective access closures, $X_Q^i \in X_C^+$. Furthermore, if any X_C^i is not indexed in $\mathcal{A}$, then Q cannot be effectively bounded since it takes no less than $\log_2 |D|$ time to even check whether X_C^i is valid in D. Therefore, for each S_i in Q, X_C^i is indexed in $\mathcal{A}$ and $X_Q^i \in X_C^+$.

$\boxed{\Leftarrow}$ Assume that for each S_i in $Q(Z)$, X_C^i is indexed in $\mathcal{A}$ and X_Q^i is in X_C^+. We give an algorithm that, given any $D \models \mathcal{A}$. computes D_Q in time independent of $|D|$, such that $Q(D_Q) = Q(D)$ and $|D_Q| \leq M$ for some positive integer M independent of $|D|$, for any $D \models \mathcal{A}$. The existence of such an algorithms verifies that Q is effectively bounded under $\mathcal{A}$.

We outline the algorithm as follows. Assume that $\Sigma_Q \vdash X_C = \bar{a}$. Given any database $D \models \mathcal{A}$, we can check whether the X_C-value $\bar{a}$ is in D in time independent of $|D|$, since X_C is indexed in $\mathcal{A}$. If not, the algorithm just returns $D_Q = \emptyset$. Otherwise, it constructs D_Q as follows. The algorithm derives a sequence of databases $D_0 \supseteq D_1 \supseteq \cdots \supseteq D_n$ such that for any $i \in [1, n]$, $D_i \subseteq D_{i-1}$, $Q(D_i) = Q(D)$, $|\pi_{X_Q^j} \sigma_{X = \bar{a}}(S_1 \times \cdots \times S_n)(D_i)| \leq N_{X_Q^j}$ for all $1 \leq j \leq i$, and D_i can be identified in time independent of $|D|$. Here $D_0 = D$. We will show how to construct D_Q from D_n and D_Q is what we need.

We show the construction by induction on i. When $i = 1$, by $D \models \mathcal{A}$ and $X_Q^1 \in X_C^+$, there exists $D_1 \subseteq D$ such that $Q(D) = Q(D_1)$ and $|\pi_{X_Q^1} \sigma_{X_C = \bar{a}}(D_1)| \leq N_{X_Q^1}$, and D_1 can be identified in $T_{X_Q^1}$ time. Suppose that when $i = k$, there exists $D_k \subseteq D$ such that $Q(D) = Q(D_k)$ and $|\pi_{X_Q^j} \sigma_{X_C = \bar{a}}(D_k)| \leq N_{X_Q^j}$ for any $j \in [1, k]$. For the inductive step, consider $i = k + 1$. Since $D_k \subseteq D$, we have that $D_k \models \mathcal{A}$. By $X_Q^{k+1} \in X_C^+$, there exists $D_{k+1} \subseteq D_k$ such that $Q(D_{k+1}) = Q(D_k) = Q(D)$, and $|\pi_{X_Q^{k+1}} \sigma_{X = \bar{a}}(D_{k+1})| \leq N_{X_Q^{k+1}}$, by taking D_k as D in the definition of effective access closures. Thus $|\pi_{X_Q^j} \sigma_{X_C = \bar{a}}(S_1 \times \cdots \times S_n)(D_k)| \leq N_{X_Q^j}$ for any $j \in [1, k+1]$ by the monotonicity of SPC queries. Hence we can construct D_{k+1}.

From this we can see that for any $i \in [1, n]$, $Q(D_i) = Q(D)$ and $|\pi_{X_Q^j} \sigma_{X = \bar{a}}(S_1 \times \cdots \times S_n)(D_i)| \leq N_{X_Q^j}$ for any $1 \leq j \leq i$, and moreover, D_i can be identified in $\Sigma_{j=1}^{i} T_{X_Q^j}$ time in i steps. Let $D_Q = \pi_{X_Q^1 \cup \cdots \cup X_Q^n} \sigma_{X_C = \bar{a}}(S_1 \times \cdots \times S_n)(D_n)$. Observe the following. (1) $Q(D) = Q(D_Q)$. (2) $|D_Q| \leq \prod_{i=1}^{n} N_{X_Q^i}$ that is independent of $|D|$. (3) The algorithm returns D_Q in time $O(\Sigma_{i=1}^{n} T_i)$, which is independent of $|D|$.

This completes the proof of Lemma 11. $\square$

Proof of Lemma 12. We show that $X \mapsto_{\mathcal{I}_E} (Y, N)$ if and only if $Y \in X^+$, when Y is a set of attributes from a renamed relation S_i.

$\boxed{\Rightarrow}$ We first show that if $X \mapsto_{\mathcal{I}_E} (Y, N)$, then $Y \in X^+$. We prove this by induction on the length of proofs $s = s_1 \circ s_2 \circ \cdots \circ s_n$ for $X \mapsto_{\mathcal{I}_E} (Y, N)$ from $\mathcal{A}$ and Q, where s_i denotes the application of one of the rules in $\mathcal{I}_E$.

(1) Basis: When $n = 1$, $X \mapsto_{\mathcal{I}_E} (Y, N)$ is deduced by a single step $s = s_1$. Then s_1 can only be the reflexivity rule or the actualization rule. By the definitions of the rules and effective access closure, it is easy to verify that $Y \in X^+$.

Algorithm compEAC

Input: Access schemas $\mathcal{A}$, an SPC query $Q(Z) = \pi_Z \sigma_C(S_1 \times \ldots \times S_n)$, and
a set X of attributes of Q.
Output: The effective access closure X^+ of X under $\mathcal{A}$ for Q.

1. $\mathit{unused} := \emptyset$; $\mathit{closure} := \{X\}$;
1. **for each** $X \to (Y, N)$ in $\mathcal{A}$ **do**
2. $\quad \mathit{unused} := \mathit{unused} \cup \bigcup_{i=1}^{n} \{S_i[X] \mapsto_{\mathcal{I}_E} (S_i[Y] \cup S_i[X], N)\}$
3. **repeat until** no further changes
4. $\quad$ **if** $W \mapsto_{\mathcal{I}_E} (Z, N) \in \mathit{unused}$, $W' \in \mathit{closure}$ and $W \subseteq W'$ **then**
5. $\quad\quad \mathit{unused} := \mathit{unused} \setminus \{W \mapsto_{\mathcal{I}_E} (Z, N)\}$;
6. $\quad\quad \mathit{closure} := \mathit{closure} \cup \{Z\}$;
7. $\quad$ **if** $\Sigma_Q \vdash Z_1 = Z_2$ and $Z_2 \subseteq Z$ and $Z \in \mathit{closure}$ and $(Z \setminus Z_2) \cup Z_1$ is indexed in $\mathcal{A}$ **then**
8. $\quad\quad \mathit{closure} := \mathit{closure} \cup \{(Z \setminus Z_2) \cup Z_1\}$
9. $\quad$ **if** $Z_1, \ldots, Z_l$ are in $\mathit{closure}$ and $Z_1 \cup \cdots \cup Z_l$ is indexed in $\mathcal{A}$ **then**
10. $\quad\quad \mathit{closure} := \mathit{closure} \cup \{Z_1 \cup \cdots \cup Z_l\}$;
11. **return** $\mathit{closure}$;

Figure 7: Algorithm compEAC for computing effective access closure

(2) Inductive step: Assume that if $X_k \mapsto_{\mathcal{I}_E} (Y_k, N_k)$ can be derived in $n = k$ steps, then $Y_K \in X_K^+$. Consider $X_{k+1} \mapsto_{\mathcal{I}} (Y_{k+1}, N_{k+1})$ derived in $k+1$ steps, via proof $s_1 \circ \cdots \circ s_{k+1}$. We show $Y_{k+1} \in X_{k+1}^+$, by considering the following cases.

(i) If s_{k+1} is the reflexivity rule or actualization rule in $\mathcal{I}_E$, then $X_{k+1} \mapsto_{\mathcal{I}_E} (Y_{k+1}, N_{k+1})$ can be derived in one step (the $k+1$-th); one can easily verify that $Y_{k+1} \in X_{k+1}^+$.

(ii) If s_{k+1} is the transitivity rule, then there exist $X \mapsto_{\mathcal{I}_E} (Y, N)$ and $Y' \mapsto_{\mathcal{I}_E} (W, N')$ that can be deduced in the first k steps via $s_1 \circ \cdots \circ s_k$, and moreover, $\Sigma_Q \vdash Y = Y'$, $X_{k+1} = X$, $Y_{k+1} = W$. By the induction hypothesis, we have $Y \in X^+$ and $W \in Y'^+$. Since $Y = Y'$, $W \in Y'^+ = Y^+ \in (X^+)^+ = X^+$. That is, $Y_{k+1} \in X_{k+1}^+$.

(iii) If s_{k+1} is the augmentation rule, then $X \mapsto_{\mathcal{I}_E} (Y, N)$ can be deduced in the first k steps via $s_1 \circ \cdots \circ s_k$, $X_{k+1} = X$ and $Y_{k+1} = X \cup Y$. By the induction hypothesis, $Y \in X^+$. That is, for any $D \models \mathcal{A}$, one can identify $D' \subseteq D$ in time independent of $|D|$ such that $Q(D) = Q(D')$ and $|\pi_Y \sigma_{X=\bar{a}}(S_1 \times \cdots \times)(D')| \leq N_Y$ for some positive integer N_Y. Thus $|\pi_{Y \cup X} \sigma_{X=\bar{a}}(S_1 \times \cdots \times S_n)(D')| \leq N_Y$. That is, $Y \cup X \in X^+$, *i.e.,* $Y_{k+1} \in X_{k+1}^+$.

(iv) If s_{k+1} is the combination rule, then $X_1 \mapsto_{\mathcal{I}_E} (Y_1, N_1)$, $\ldots$, $X_l \mapsto_{\mathcal{I}} (Y_l, N_l)$ can be deduced within k steps, via proof $s_1 \circ s_2 \cdots \circ s_k$, while $Y_{k+1} = Y_1 \cup \cdots \cup Y_k$ and $X_{k+1} = X_1 \cup \cdots \cup X_k$. By the inductive hypothesis, $Y_i \in X_i^+$ for $i \in [1, l]$. That is, for any database $D \models \mathcal{A}$, one can identify $D'_i \subseteq D$ in time independent of $|D|$ such that $|\pi_{Y_i} \sigma_{X_i = \bar{a}}(S_1 \times \cdots \times S_n)(D'_i)| \leq N_{Y_i}$ and $Q(D) = Q(D'_i)$. Along the same lines as the proof of Lemma 11, one can show that there exists $D_Q \subseteq D'_i$ for each $i \in [1, n]$ such that $Q(D_Q) = Q(D)$ and $\pi_{Y_1 \cup \cdots \cup Y_n} \sigma_{X_1 \cup \cdots \cup X_n = \bar{a}}(S_1 \times \cdots \times S_n)(D_Q) \leq \prod_{i=1}^{n} N_{Y_i}$. Since $Y_1 \cup \cdots \cup Y_n$ is indexed, $Y_1 \cup \cdots \cup Y_n \in (X_1 \cup \cdots \cup X_n)^+$. That is, $Y_{k+1} \in X_{k+1}^+$.

(v) When s_{k+1} is X-equality or s_{k+1} is Y-equality, the argument is similar to its counterpart given in the proof of Lemma 10.

Therefore, the statement holds for $k+1$.

$\boxed{\Leftarrow}$ We next show that if $Y \subseteq X^+$, then $X \mapsto_{\mathcal{I}_E} (Y, N)$ for some positive integer N. We first show that for any set X of variables, there exists a proof s that entails $X \mapsto_{\mathcal{I}_E} (X^+, N)$. Since $Y \subseteq X^+$, we then have a proof $s \circ \varphi \circ \varphi'$ that entails $X \mapsto_{\mathcal{I}_E} (Y, N)$, where φ and φ' are the reflexivity rule and the transitivity rule, respectively.

To show that $X \mapsto_{\mathcal{I}_E} (X^+, N)$ for some N, we first provide algorithm compEAC shown in Fig. 7 that, given any access schema $\mathcal{A}$, any SPC query Q and a set X of attributes of Q, computes X^+ under $\mathcal{A}$ for Q.

One can verify that compEAC correctly computes X^+, by induction on the steps of fetch and chkComb of getting D_Q from D for Q based on the specification of effective boundedness of Q.

Leveraging algorithm compEAC, we show that $X \mapsto_{\mathcal{I}_E} (X^+, N)$ for some N. Let $\mathit{closure}_i$ be the value of variable $\mathit{closure}$ after i iterations for some execution on input $\mathcal{A}$, Q and X. We prove that $X \mapsto_{\mathcal{I}} (X^+, N)$ by induction on iteration step i of compEAC (steps 4-10). Initially, we set $\mathit{closure}_0 = \{X\}$.

(1) Basis: $X \mapsto_{\mathcal{I}_E} (\mathit{closure}_0, N_0)$ follows from the reflexivity rule.

(2) Inductive step: Suppose that a proof $s_{k_i} = s_1 \circ \cdots \circ s_{k_i}$ for $X \mapsto_{\mathcal{I}_E} (\mathit{closure}_i, N_i)$ has been constructed for some N_i. We show $X \mapsto_{\mathcal{I}_E} (\mathit{closure}_{i+1}, N_{i+1})$ by considering the following three cases.

Case (1) Suppose $W \mapsto_{\mathcal{I}} (V, N)$ is chosen for the $(i+1)^{st}$ step. It follows that there exists $W' \in \mathit{closure}_i$, $W \subset W'$, $\mathit{closure}_{i+1} = \mathit{closure}_i \cup V$. Then extend the proof s_{k_i} by adding the following steps:

∘ let s_{k_i+1} be the reflexivity rule, leading to $W' \mapsto_{\mathcal{I}_E} (W, 1)$;
∘ let s_{k_i+2} be the transitivity rule, leading to $W' \mapsto_{\mathcal{I}_E} (B, N)$ for some N;
∘ let s_{k_i+3} be the reflexivity rule, leading to $closure_i \mapsto_{\mathcal{I}_E} (W', 1)$;
∘ let s_{k_i+4} be the transitivity rule, leading to $closure_i \mapsto_{\mathcal{I}_E} (V, N)$;
∘ let s_{k_i+5} be the augmentation rule, leading to $closure_i \mapsto_{\mathcal{I}_E} (closure_{i+1}, N_1)$;
∘ let s_{k_i+6} be the transitivity rule, leading to $X \mapsto_{\mathcal{I}_E} (closure_{i+1}, N_2)$.

Case (2) Suppose that steps 7-8 correspond to the $(i+1)^{st}$ step. It follows that $closure_{i+1} = closure_i \cup \{(V \setminus V_2) \cup V_1\}$. Then extend the proof s_{k_i} by adding the following steps:
∘ let s_{k_i+1} be the reflexivity rule, leading to $closure_i \mapsto_{\mathcal{I}_E} (V, 1)$;
∘ let s_{k_i+2} be the Y-equality rule, leading to $closure_i \mapsto_{\mathcal{I}_E} ((V \setminus V_2) \cup V_1, 1)$;
∘ let s_{k_i+3} be the augmentation rule, leading to $closure_i \mapsto_{\mathcal{I}_E} (closure_i \cup \{(V \setminus V_2) \cup V_1\}, 1)$;
∘ let s_{k_i+4} be the transitivity rule, leading to $X \mapsto_{\mathcal{I}_E} (closure_{i+1}, N)$.

Case (3) Suppose that steps 9-10 correspond to the $(i+1)^{st}$ step. It follows that $closure_{i+1} = closure_i \cup \{V_1 \cup \cdots \cup V_l$. Then extend the proof s_{k_i} by adding the following steps:
∘ let s_{k_i+1} be the reflexivity rule, leading to $closure_i \mapsto_{\mathcal{I}_E} (V_1, 1)$;
... ...
∘ let s_{k_i+l} be the reflexivity rule, leading to $closure_i \mapsto_{\mathcal{I}_E} (V_l, 1)$;
∘ let s_{k_i+l+1} be the combination rule, leading to $closure_i \mapsto_{\mathcal{I}_E} (V_1 \cup \cdots \cup V_l, 1)$;
∘ let s_{k_i+l+2} be the transitivity rule, leading to $X \mapsto_{\mathcal{I}_E} (V_1 \cup \cdots \cup V_l, 1)$.

Hence the statement holds for $i+1$.

This completes the proof of Lemma 12 and Theorem 4. □

Appendix B: Proofs of Complexity and Approximation

Nontrivial parameters. We say that a set X_P of parameters are *trivial* to Q, if it covers *almost all* attributes in Q: the number of attributes of Q that are not in X_P is a constant. Intuitively, if we instantiating a set of trivial parameters, then the "size" of Q is constant: it can be determined in $O(1)$ time whether Q is effectively bounded. A set of X_P is called *nontrivial* to Q, if it is not trivial.

We are interested in *nontrivial* parameters. For the convenience of discussion, we assume *w.l.o.g.* that, there exists a number $\alpha \in (0, 1]$ such that, for any SPC query Q and any set X_P of nontrivial parameters of Q, $\frac{|X_P|}{|Q|} \leq 1 - \alpha$, where $|Q|$ is number of attributes in Q. That is, there are at least $\alpha|Q|$ attributes that are not instantiated in X_P.

Proof of Theorem 7

Proof of Theorem 7(1): NP-completeness.

We next show that $\mathsf{DP}(Q, \mathcal{A})$ is NP-complete when considering nontrivial parameters. We first show it is in NP, and then show it is NP-hard.

Upper bound. For the NP-hard upper bound it suffices to observe that the following NP algorithm correctly decides whether there exists a set of dominating parameters of Q under $\mathcal{A}$.

Given an SPC query Q and access schema $\mathcal{A}$, the algorithm simply guesses a set X of parameters of Q and check whether X_P is nontrivial and whether $Q(X = \bar{a})$ is effectively bounded under $\mathcal{A}$ for all given X_P value $\bar{a}$, and returns "yes" if it is. The algorithm returns "no" when all sets of parameters of Q are checked. Note that the checking is in $O(|Q|(|Q| + |\mathcal{A}|))$ time as shown by Theorem 6. Thus, $\mathsf{DP}(Q, \mathcal{A})$ is in NP.

Lower bound. The NP-lower bound is established by a reduction from the 3SAT problem. An instance of the 3SAT problem is a formula $\varphi = C_1 \wedge \cdots \wedge C_r$ with $C_j = \ell_1^j \vee \ell_2^j \vee \ell_3^j$ and for $k \in \{1, 2, 3\}$ and $j \in [1, r]$, ℓ_k^j is either a variable or a complement of a variable from $X = \{x_1, \ldots, x_n\}$, and is to determine whether φ is satisfiable. This problem is known to be NP-complete.

Given an instance of the 3SAT problem, *i.e.,* a formula φ described above, we define an instance of $\mathsf{DP}(Q, \mathcal{A})$, *i.e.,* a relational schema $\mathcal{R}$, an SPC query Q of $\mathcal{R}$ and an access schema $\mathcal{A}$ of $\mathcal{R}$, such that there exists a truth assignment μ_X such that $\mu_X(\varphi)$ is *true* if and only if there exists a set X_P of parameters such that $Q(X_P = \bar{a})$ is effectively bounded for all X_P values $\bar{a}$.

More specifically, we define the following.

(1) The relational schema $\mathcal{R}$ consists of four relation schemas $R_X(X_1, X_2)$ of 2-arity, $R_C(C_1, \ldots, C_n)$ of n-arity, $R_D(W_1, \ldots, W_{2n}, V_1^1, \ldots, V_n^1, \ldots, V_1^r, \ldots, V_n^r)$ of $(2n + nr)$-arity, $R_Y(Y_1^1, \ldots, Y_n^1, Y_1^2, \ldots, Y_n^2, Y_1^3, \ldots, Y_n^3, Y_1^4, \ldots, Y_n^4, U_1^1, \ldots, U_n^r, \ldots, U_1^r, \ldots, U_n^r)$ of $(4n+nr)$-arity, and $R_A(A)$ of 1-arity. Intuitively, (a) R_X is to encode variables in X of φ; (b) R_C is to encode truth assignment of clauses in φ; and (c) R_D is to connect R_X and R_C; The use of R_Y and R_A will be explained below.

(2) The SPC query $Q = \sigma_C(S_1^X \times \cdots \times S_n^X \times S^Y \times S^D \times S_1^C \times \cdots S_r^C \times S_1^A \times \cdots \times S^A_{\frac{(8n+3nr)(1-\alpha)-n}{\alpha}}$, where

(a) $S_1^X, \ldots, S_n^X$ are renamings of relation schema R_X; S^Y is a renaming of relation schema R_Y; S^D is a renaming of relation schema R_D; $S_1^C, \ldots, S_r^C$ are renamings of relation schema R_C; and $S_1^A, \ldots, S^A_{\frac{(8n+3nr)(1-\alpha)-n}{\alpha}}$ are renamings of relation schema R_A;

(b) σ_C consists of the following equality atoms.

- For each clause $C_j = \ell_1^j \vee \ell_2^j \vee \ell_3^j$, $\mathsf{enX}(\ell_i^j) = \mathsf{enDW}(\ell_i^j)$, and $\mathsf{enDV}(\ell_i^j) = \mathsf{enC}(\ell_i^j)$, for all $i \in \{1,2,3\}$ and $j \in [1,r]$, are included in σ_C, in which $\mathsf{enC}(\ell_i^j)$ is $S_j^C[C_1, \ldots, C_n]$, $\mathsf{enDV}(\ell_i^j)$ is $S^D[V_1^j, \ldots, V_n^j]$, $\mathsf{enX}(\ell_i^j)$ and $\mathsf{enDW}(\ell_i^j)$ are $S_k^X[X_1]$ and $S^D[W_{2k-1}]$, respectively, if ℓ_i^j is x_k, and are $S_k^X[X_2]$ and $S^D[W_{2k}]$, respectively, if ℓ_i^j is $\bar{x}_k$, for any $k \in [1,n]$. That is, whenever whenever a
- for each $i \in [1,n]$, $S_i^X[X_1] = S^Y[Y_i^1]$ and $S_i^X[X_2] = S^Y[Y_i^2]$ are in σ_C;
- for each $i \in [1,n]$, $S^Y[Y_i^3] = S^D[W_{2i-1}]$ and $S^Y[Y_i^4] = S^D[W_{2i}]$ are in σ_C of Q; and
- for each $i \in [1,n]$, $j \in [1,r]$, $S^D[V_i^j] = S^Y[U_i^j]$.

(3) The access schema $\mathcal{A}$ is defined as follows.

(a) For $R_X(X_1, X_2)$, we have $X_i \to (X_i, 1)$ ($i \in \{1,2\}$) and $(X_1, X_2) \to ((X_1, X_2), 1)$ in $\mathcal{A}$.

(b) For $R_C(C_1, \ldots, C_n)$, we include $(C_1, \ldots, C_n) \to ((C_1, \ldots, C_n), 1)$ in $\mathcal{A}$.

(c) For $R_A(A)$, we include $A \to (A, 1)$ in $\mathcal{A}$.

(d) For $R_Y(Y_1^1, \ldots, Y_n^1, Y_1^2, \ldots, Y_n^2, Y_1^3, \ldots, Y_n^3, Y_1^4, \ldots, Y_n^4, U_1^1, \ldots, U_n^r, \ldots, U_1^r, \ldots, U_n^r)$, we include the following in $\mathcal{A}$:

- $Y_i^1 \to ((Y_i^3, Y_i^4), 1)$, for all $i \in [1,n]$;
- $(Y_i^2 \to ((Y_i^3, Y_i^4), 1)$ in $\mathcal{A}$, for all $i \in [1,n]$;
- $\mathsf{attr}(R_Y) \to (\mathsf{attr}(R_Y), 1)$; and
- $\bigcup\limits_{i=1}^{n} \bigcup\limits_{j=1}^{r} (Y_i^3 \cup Y_i^4 \cup U_i^j) \to (\mathsf{attr}(R_Y), 1)$.

(e) For R_D, we include the following in $\mathcal{A}$:

- $\mathsf{attr}(R_D) \to (\mathsf{attr}(R_D), 1)$; and
- for each clause $C_j = \ell_1^j \vee \ell_2^j \vee \ell_3^j$, $\mathcal{A}$ also includes $\mathsf{enDW}(\ell_i^j) \to (\mathsf{enDV}(\ell_i^j), 1)$, for each $i \in \{1,2,3\}$.

Intuitively, (i) access constraints on R_X is to ensure that the encoding of any truth assignment of variables in X for φ is indexed; (ii) access constraints on R_C is to ensure each encoding of truth value of clauses C_j ($j \in [1,r]$) is indexed; (iii) access constraints on R_A is to ensure that when R_A is instantiated, it will still possibly be effectively bounded by guaranteeing the indexing condition; and (iv) access constraints on R_Y ensures that, whenever 1) $R_i^X[X_1]$ or $R_i^X[X_2]$ are instantiated for all $i \in [1,n]$ and 2) $R^D[\bigcup_{i=1}^n \bigcup_{j=1}^r V_i^j]$ can be deduced via $\mathcal{I}_E$, then together with σ_C of Q, R_i^X and $R^D[W_1, \ldots, W_{2n}]$ can be deduced with rules in $\mathcal{I}_E$, for all $i \in [1,n]$. That is, access constraints on R_Y ensures that S^Y is indexed if and only if the truth assignment encoded by S_i^X's makes φ *true* and moreover, each of the variables x_i in X of φ is assigned a truth value.

We next show that there exists a truth assignment to φ if and only if there exists a nontrivial set X_P of dominating parameters of Q under $\mathcal{A}$.

$\boxed{\Rightarrow}$ Assume there exists a truth assignment μ_X to variables in X such that $\mu_X(\varphi)$ is *true*. Define X_P as follows: (1) for each $i \in [1,n]$, if $\mu_X(x_i)$ is *true*, then X_P includes $S_i^X[X_1]$; otherwise X_P includes $S_i^X[X_2]$; and (2) X_P also includes $S_1^A[A]$, $\ldots$, $S^A_{\frac{(8n+3nr)(1-\alpha)-n}{\alpha}}[A]$. Note that $|X_P|/|Q| = \dfrac{\dfrac{(8n+3nr)(1-\alpha)-n}{\alpha} + n}{\dfrac{(8n+3nr)(1-\alpha)-n}{\alpha} + 8n + 3nr}$ = 1 - α, thus X_P is nontrivial. Observe that, by reflexivity, transitivity and combination rule, we have $X_P \mapsto_{\mathcal{I}_E} (\bigcup_{i=1}^r S_i^C[C_1, \ldots, C_n] \cup \bigcup_{i=1}^{\frac{(8n+3nr)(1-\alpha)-n}{\alpha}} S_i^A[A] \cup \mathsf{attr}(S^Y), 1)$. By access constraint $\bigcup_{i=1}^n \bigcup_{j=1}^r (Y_i^3 \cup Y_i^4 \cup U_i^j) \to (\mathsf{attr}(S^Y), 1)$ and equations in σ_C, we can further have that $X_P \mapsto_{\mathcal{I}_E} (\bigcup_{i=1}^r S_i^C[C_1, \ldots, C_n] \cup \bigcup_{i=1}^{\frac{(8n+3nr)(1-\alpha)-n}{\alpha}} S_i^A[A] \cup \mathsf{attr}(S^Y) \cup \mathsf{attr}(S^D), 1)$. That is, $X_P \mapsto_{\mathcal{I}_E} (\bigcup_{i=1} X_Q^i,\ 1)$. By Theorem 4, we know that $Q(X_P = \bar{a})$ is effectively bounded under $\mathcal{A}$ for any X_P values $\bar{a}$.

$\boxed{\Leftarrow}$ Assume that there exists a nontrivial set X_P of parameters of Q such that $Q(X_P = \bar{a})$ is effectively bounded under $\mathcal{A}$ for any X_P values $\bar{a}$. by Theorem 4 and the definition of access schema $\mathcal{A}$, X_P must contain $S_i^A[A]$ for all $i \in [1, \frac{(8n+3nr)(1-\alpha)-n}{\alpha}]$ since otherwise there exists some $S_i^A[A]$ that cannot be entailed from X_P via $\mathcal{I}_E$. Observe that X_P can only contain at most another n attributes since it is nontrivial. Further more, note that (a) if X_P does not instantiate at least one attribute from $S_1^X, \ldots, S_n^X$, or (b) if $S_1^C, \ldots, S_r^C$ cannot deduced from X_P via $\mathcal{I}_E$, then S^Y can not be deduced from X_P via $\mathcal{I}_E$. Thus, by the definition of Q, one can verify that for each $i \in [1,n]$, either $S_i^X[X_1]$ or S_i^X is instantiated directly by X_P, but not both. Here we say that X_P instantiate attribute A if after instantiating X_P with $\bar{a}$, $\Sigma_Q \vdash x = c$ for some constant c. That is, X_P encodes a truth assignment μ_X for X in φ: for each x_k in X, if $S_k^X[X_1] \in X_P'$, then $\mu_X(x_k) = $ *true*; if $S_k^X[X_2] \in X_P'$, then

$\mu_X(x_k) = \textit{false}$, such that $\mu_X(\varphi) = \mu_X(C_1) \wedge \cdots \wedge \mu_X(C_r) = \textit{true}$.

Proof of Theorem 7(2): NPO**-completeness**.

We next show that $\mathsf{MDP}(Q, \mathcal{A})$ is NPO-complete. Since the corresponding decision problem of $\mathsf{MDP}(Q, \mathcal{A})$ is in NP, $\mathsf{MDP}(Q, \mathcal{A})$ is an NPO problem. We just need to show it is NPO-hard.

We show $\mathsf{MDP}(Q, \mathcal{A})$ is NPO-hardness by a PTAS-reduction from the MINIMUM WEIGHTED 3SAT problem (MW3SAT), which is known NPO-complete (cf. [12]).

Let P_1 and P_2 are two optimization problems in NPO. P_1 is said to be PTAS-reducible to P_2, in symbols $P_1 \leq_{PTAS} P_2$, if three functions f, g and c exist such that:

(i) For any instance $x \in I_{P_1}$ and for any rational $r > 1$, $f(x, r) \in I_{P_2}$ is computable in time polynomial with respect to $|x|$.

(ii) For any instance $x \in I_{P_1}$, for any rational $r > 1$, and for any $y \in SOL_{P_2}(f(x, r))$, $g(x, y, r) \in SOL_{P_1}(x)$ is computable in time polynomial *w.r.t.* both $|x|$ and $|y|$.

(iii) $c : (1, +\infty) \to (1, +\infty)$.

(iv) For any instance $x \in I_{P_1}$, for any rational $r > 1$, and for any $y \in SOL_{P_2}(f(x, r))$, $R_{P_2}(f(x, r), y) \leq c(r)$ implies $R_{P_1}(x, g(x, y, r)) \leq r$.

Here the triple (f, g, c) is said to be an PTAS-reduction from P_1 to P_2 (note that functions f and g depend on r). It is know that, if there exists an PTAS-reduction from P_1 to P_2, and P_1 is NPO-hard, then P_2 is also NPO-hard [12].

An instance of the MW3SAT problem is a well-formed Boolean formula $\varphi = C_1 \wedge \cdots \wedge C_r$ with variables $x_1, \ldots, x_n$ of nonnegative weights $w_1, \ldots, w_n$, where each C_j is of the form $\ell_1^j \vee \ell_2^j \vee \ell_3^j$, for any $j \in [1, r]$. The problem is to find a truth assignment μ to variables $x_1, \ldots, x_n$ that satisfies φ, such that $\sum_{i=1}^n w_i \mu(x_i)$ is minimized. Here Boolean values *true* and *false* are identified with 1 and 0, respectively.

We next present a PTAS-reduction (f, g, c) from MW3SAT to $\mathsf{MDP}(Q, \mathcal{A})$.

<u>*function f*</u>. Given any instance φ of MAXIMUM WEIGHTED 3SAT, and any rational number $r > 1$, f constructs an instance $f(\varphi, r)$ of $\mathsf{MDP}(Q, \mathcal{A})$ as follows.

(1) If $r > \frac{\sum_{i=1}^n w_i}{\min\{w_i | i \in [1,n]\}}$, then construct the same instance of $\mathsf{MDP}(Q, \mathcal{A})$ as in the proof above.

(2) If $1 \leq r \leq \frac{\sum_{i=1}^n w_i}{\min\{w_i | i \in [1,n]\}}$, then construct the an instance of $\mathsf{MDP}(Q, \mathcal{A})$ that is the same to the instance used above, except that $\mathcal{A}$ is set to $\emptyset$.

Intuitively, (a) when $r > \frac{\sum_{i=1}^n w_i}{\min\{w_i | i \in [1,n]\}}$, $SOL_{\mathsf{MDP}}(Q, \mathcal{A}) \neq \emptyset$ if and only if $SOL_{MW3\mathsf{SAT}}(\varphi) \neq \emptyset$. Furthermore, note that $R_{\mathsf{MDP}}(Q, \mathcal{A})$ is always 1 since when there exists nontrivial X_P that makes Q effectively bounded, the minimum X_P is exactly the one constructed in the proof above, which consists of $S_1^A[A], \ldots, S^A_{(\frac{1}{\alpha}-1)(n+r+1)}[A]$, and the m attributes selected from $S_1^X, \ldots, S_n^X$, where m is the number of distinct literals in φ. (b) When $1 \leq r \leq \frac{\sum_{i=1}^n w_i}{\min\{w_i | i \in [1,n]\}}$, $SOL_{\mathsf{MDP}}(Q, \mathcal{A})$ is $\emptyset$ since Q can never be made effectively bounded under $\mathcal{A} = \emptyset$.

<u>*function g*</u>. For any rational $r > 1$, and any feasible solution X_P of $SOL_{\mathsf{MDP}}(Q, \mathcal{A})$, where Q and $\mathcal{A}$ is constructed by $f(\varphi, r)$ above, we define $g(x, X_P, r)$ to be the corresponding truth assignment μ_X that corresponds to X_P for Q and $\mathcal{A}$ as given in the proof above. Note that, $\sum_{i=1}^n w_i \mu_X(x_i) \leq \sum_{i=1}^n w_i$.

<u>*function c*</u>. We define $c(r) = r$ for any $r > 1$.

We next show that (f, g, c) is indeed a PTAS-reduction, by verifying that (f, g, c) satisfies the above four conditions of PTAS-reductions. By the definition of f, g and c, we know that conditions (i), (ii) and (iii) are satisfied. Note that, for any instance φ of MW3SAT, for any rational $r > 1$, if there exists any feasible solution, *i.e.*, a nontrivial set X_P of dominating parameters, to instance $f(\varphi, r)$ of $\mathsf{MDP}(Q, \mathcal{A})$, then we must have $r > \frac{\sum_{i=1}^n w_i}{\min\{w_i | i \in [1,n]\}}$. Thus, the performance ratio of $\mathsf{MDP}(Q, \mathcal{A})$ $R_{\mathsf{MDP}}(Q, \mathcal{A}) = 1 \leq r = c(r)$, and the performance ratio of MW3SAT $R_{mw3\mathsf{SAT}}(\varphi) \leq \frac{\sum_{i=1}^n w_i}{\min\{w_i | i \in [1,n]\}} < r$. That is, condition ($iv$) is established. Thus, (f, g, c) is a PTAS-reduction from MW3SAT to $\mathsf{MDP}(Q, \mathcal{A})$. Since that former is NPO-complete, so is $\mathsf{MDP}(Q, \mathcal{A})$.

Proof of Theorem 8

We first extend $\mathcal{I}_B$ and $\mathcal{I}_E$ to $\mathcal{I}_B^M$ and $\mathcal{I}_E^M$ to cope with the cases where M is part of the input, as shown in Fig. 8 and Fig. 9.

Follow the same lines as the proofs of Theorem 3 and Theorem 4, one can verify the following Lemma.

Lemma 13: *Consider any* SPC *query $Q(Z)$, any access schema $\mathcal{A}$.*

(1) $Q(Z)$ is M-bounded under $\mathcal{A}$ if and only if $X_B \cup X_C \mapsto_{\mathcal{I}_B^M} (X_B \cup Z, M)$.

(2) $Q(Z)$ is effectively M-bounded under $\mathcal{A}$ if and only if

- *whether $\bigcup_{i=1}^n X_Q^i$ is indexed in $\mathcal{A}$; and*
- *whether $X_C \mapsto_{\mathcal{I}_E^M} (\bigcup_{i=1}^n X_Q^i, M)$.*

(3) For any $X \mapsto_{\mathcal{I}_B^M} (Y, M)$ (resp. $X \mapsto_{\mathcal{I}_E^M} (Y, M)$) and $X \subseteq Y$, there exists a proof no longer than $O(|Q|(|Q| + |\mathcal{A}|))$ that

(**Reflexivity**) If $X' \subseteq X$, then $X \mapsto_{\mathcal{I}_B^M} (X', 1)$.
(**Relaxiation**) If $X \mapsto_{\mathcal{I}_B^M} (Y, N)$, then $X \mapsto_{\mathcal{I}_B^M} (Y, N')$ for any $N' \geq N$.
(**Actualization**) If $X \to (Y, N)$ is in $\mathcal{A}$, then $S_i[X] \mapsto_{\mathcal{I}_B^M} (S_i[Y], N)$ for each i in $[1, n]$.
(**Augmentation**) If $X \mapsto_{\mathcal{I}_B^M} (Y, N)$, then $X \cup W \mapsto_{\mathcal{I}_B^M} (Y \cup W, N)$.
(**Transitivity**) If $X \mapsto_{\mathcal{I}_B^M} (Y_1, N_1)$, $Y_2 \mapsto_{\mathcal{I}_B^M} (W, N_2)$, and $\Sigma_Q \vdash Y_1 = Y_2$, then $X \mapsto_{\mathcal{I}_B^M} (W, N_1 * N_2)$.
(X-**equality**) If $\Sigma_Q \vdash X' = X''$, $X'' \subseteq X$, and $X \mapsto_{\mathcal{I}_B^M} (Y, N)$, then $(X \setminus X'') \cup X' \mapsto_{\mathcal{I}_B^M} (Y, N)$.
(Y-**equality**) If $\Sigma_Q \vdash Y' = Y''$, $Y'' \subseteq Y$, and $X \mapsto_{\mathcal{I}_B^M} (Y, N)$, then $X \mapsto_{\mathcal{I}_B^M} (Y \cup Y', N)$.
(**Inner-Aggregation**) If $X \mapsto_{\mathcal{I}_B^M} (Y, N)$ $X' \mapsto_{\mathcal{I}_B^M} (Y', N')$, and Y and Y' are both from some S_i of Q, then $X \cup X' \mapsto_{\mathcal{I}_B^M} (Y \cup Y', N * N')$.
(**Cross-Aggregation**) If $X \mapsto_{\mathcal{I}_B^M} (Y, N)$ $X' \mapsto_{\mathcal{I}_B^M} (Y', N')$, and Y and Y' are both from two distinct S_i of Q, then $X \cup X' \mapsto_{\mathcal{I}_B^M} (Y \cup Y', N + N')$.

Figure 8: Deduction rules $\mathcal{I}_B^M$ for M-boundedness

entails it. □

Proof of Theorem 8(1): NP-completeness.

We first show the problem is NP-hard, and then show it is in NP.

Lower bound. The NP-lower bound is established by a reduction from the VERTEX COVER problem. An instance of the VERTEX COVER problem consists of an undirected graph $G(V, E)$ and an integer K, and is to determine whether there exists a set $V' \subset V$ such that $|V'| \leq K$.

Given an instance of the VERTEX COVER problem described above, we construct an instance of $\mathsf{Bnd}(Q, \mathcal{A}, M)$ such that there exists the set V' if and only if Q is M-bounded under $\mathcal{A}$.

The instance of $\mathsf{Bnd}(Q, \mathcal{A}, M)$ is defined as follows.

(1) The relational schema $\mathcal{R}$ consists of only one relation schema $R(V_1, \ldots, V_{|V|}, U_1, \ldots, U_{|E|})$ of $|V| + |E|$-arity, Intuitively, $R[V_i]$ is to encode the node v_i in V of G, $R[U_j]$ is to encode edge e_j in E of G, for any $i \in [1, |V|]$ and $j \in [1, |E|]$.

(2) The SPC query Q is defined as $Q = \pi_{U_1, \ldots U_{|E|}} \sigma_C(R)$, where σ_C consists of the following: for each $i \in [1, |V|]$, $R[V_i] = c$ is in σ_C, in which c is a constant; and

(3) The access schema $\mathcal{A}$ includes the following: for each $i \in [1, |V|]$, include $V_i \to (\mathcal{U}_i, N_0)$ in $\mathcal{A}$, where $\mathcal{U}_i$ is the set of all those attributes U_j of R such that node v_i in V of G is a node in edge e_j in E of G, where v_i and e_j are corresponding node and edge that encoded by V_i and U_j.

(4) Let $M = N_0^K$.

We next show that there exists a set $V' \subset V$ such that V' covers E and $|V'| \leq K$ if and only if Q is M-bounded under $\mathcal{A}$.

$\boxed{\Rightarrow}$ Assume there exists a set $V' = \{v_{k_1}, \ldots, v_{k_K}\}$ covers E of G, where $k_j \in [1, n]$. Note that $X_C = \{V_1, \ldots, V_{|V|}\}$, $X_B = \emptyset$, and $Z = \{U_1, \ldots, U_{|E|}\}$. Let X'_C be the subset of X_C such that each attribute in X'_C corresponds to a node in V'. Therefore, $X'_C \cup X_B \mapsto_{\mathcal{I}_B^M} (\mathcal{U}_i, N_0)$ for each $i \in \{k_1, \ldots, k_K\}$, where $\mathcal{U}_i$ is the set of all attributes in R that encode edges e in E that contain node v_i in V' as a node. Since V' covers E of G, $Z = \bigcup_{j=1}^{K} \mathcal{U}_{j_k}$. Thus $X_C \cup X_B \mapsto_{\mathcal{I}_B^M} (X_B \cup Z, M)$. By Lemma 13, we have that Q is M-bounded.

$\boxed{\Leftarrow}$ Assume that Q is M-bounded. By the the rules in $\mathcal{I}_B^M$ and Lemma 13, there must exists a set $\mathcal{V} = \{V_{i_1}, \ldots, V_{i_K}\}$ of attributes of R, where $i_k \in [1, n]$ for all $k \in [1, K]$, such that $V_{i_k} \mapsto_{\mathcal{I}_B^M} (\mathcal{U}_{i_k}, N_0)$ and moreover, $\bigcup_{k=1}^{K} \mathcal{U}_{i_k} = Z = \bigcup_{i=1}^{|E|} U_i$. Let V' be the set consisting of all nodes encoded by attributes in $\mathcal{V}$. By the analysis above, V' covers all edges in E of G, and V' contains no more than K nodes.

Upper bound. We show $\mathsf{Bnd}(Q, \mathcal{A}, M)$ is in NP by an NP algorithm that works as follows: first guess a proof of length bounded by $O(|Q|^2 |\mathcal{A}|^3))$ with $\mathcal{I}_B^M$, and then verify it and check whether it leads to $X_C \cup X_B \mapsto_{\mathcal{I}_B^M} (X_B \cup Z, M)$. Note that the verification and checking are both in PTIME. Thus it is an NP algorithm.

Proof of Theorem 8(2): NP-completeness.

We first show it is NP-hard, and then show it is in NP.

Lower Bound. The NP-lower bound is established by a reduction from the VERTEX COVER problem, which is a revision of the one used in the proof of Theorem 8(1) above.

Given an instance of the VERTEX COVER problem, we construct an instance of $\mathsf{Bnd}(Q, \mathcal{A}, M)$ such that there exists the set V' if and only if Q is M-bounded under $\mathcal{A}$.

The instance of $\mathsf{Bnd}(Q, \mathcal{A}, M)$ is defined as follows.

(1) The relational schema $\mathcal{R}$ and Q are the same to their counterparts used above (proof of Theorem 8(1)).

(**Reflexivity**) If $X' \subseteq X$, then $X \mapsto_{\mathcal{I}_E^M} (X', 1)$.
(**Relaxiation**) If $X \mapsto_{\mathcal{I}_E^M} (Y, N)$, then $X \mapsto_{\mathcal{I}_E^M} (Y, N')$ for any $N' \geq N$.
(**Actualization**) If $X \to (Y, N)$ is in $\mathcal{A}$, then $S_i[X] \mapsto_{\mathcal{I}_E^M} (S_i[Y], N)$ for each i in $[1, n]$.
(**Transitivity**) If $X \mapsto_{\mathcal{I}_E^M} (Y, N)$ and $Y \mapsto_{\mathcal{I}_E^M} (W, N')$, then $X \mapsto_{\mathcal{I}_E^M} (W, N * N')$.
(**Augmentation**) If $X \mapsto_{\mathcal{I}_E^M} (Y, N)$ and $X \cup Y$ is *indexed*, then $X \mapsto_{\mathcal{I}_E^M} (X \cup Y, N)$.
(**Inner-Combination**) If $X_1 \mapsto_{\mathcal{I}_E^M} (Y_1, N_1), \ldots, X_k \mapsto_{\mathcal{I}_E^M} (Y_k, N_k)$, $\Sigma_Q \vdash Y_1 = Y_1', \ldots, \Sigma_Q \vdash Y_k = Y_k'$,
$\bigcup_{i=1}^k (X_i \cup Y_i' \cup Y_i)$ is *indexed* in $\mathcal{A}$,
$\bigcup_{i=1}^k X_i$ consists of attributes from the same renaming of R in Q only, and
$\bigcup_{i=1}^k Y_i$ consists of attributes from the same renaming of R in Q only, then
$X_1 \cup \cdots \cup X_k \mapsto_{\mathcal{I}_E^M} (Y_1' \cup \cdots \cup Y_k', N_1 * \cdots * N_k)$.
(**Cross-Combination**) If $X_1 \mapsto_{\mathcal{I}_E^M} (Y_1, N_1), \ldots, X_k \mapsto_{\mathcal{I}_E^M} (Y_k, N_k)$, $\Sigma_Q \vdash Y_1 = Y_1', \ldots, \Sigma_Q \vdash Y_k = Y_k'$, and
$\bigcup_{i=1}^k (X_i \cup Y_i' \cup Y_i)$ is *indexed* in $\mathcal{A}$,
$\bigcup_{i=1}^k X_i$ consists of attributes from the distinct renamings of R in Q, and
$\bigcup_{i=1}^k Y_i$ consists of attributes from the distinct renamings of R in Q, then
$X_1 \cup \cdots \cup X_k \mapsto_{\mathcal{I}_E^M} (Y_1' \cup \cdots \cup Y_k', N_1 + \cdots + N_k)$.
(X-**Equality**) If $\Sigma_Q \vdash X' = X''$, $X'' \subseteq X$, $X \mapsto_{\mathcal{I}_E^M} (Y, N)$, and $X' \cup X \cup Y$ is *indexed* in $\mathcal{A}$, then
$(X \setminus X'') \cup X' \mapsto_{\mathcal{I}_E^M} (Y, N)$.
(Y-**Equality**) If $\Sigma_Q \vdash Y' = Y''$, $Y'' \subseteq Y$, $X \mapsto_{\mathcal{I}_E^M} (Y, N)$, and $X \cup Y \cup Y'$ is *indexed* in $\mathcal{A}$, then
$X \mapsto_{\mathcal{I}_E^M} (Y \cup Y', N)$.

Figure 9: Rules $\mathcal{I}_E^M$ for effective M-boundedness

(2) The access schema $\mathcal{A}$ includes the following:

- for each $i \in [1, |V|]$, include $V_i \to (\mathcal{U}_i, N_0)$ in $\mathcal{A}$, where $\mathcal{U}_i$ is the set of all those attributes U_j of R such that node v_i in V of G is a node in edge e_j in E of G, where v_i and e_j are corresponding node and edge that encoded by V_i and U_j;
- for each $i \in [1, |V|]$, include $V_i \to (V_i, 1)$;
- $\mathcal{A}$ also includes $(V_1, \ldots, V_{|V|}) \to ((U_1, \ldots, U_{|E|}), 1)$.

(3) Let $M = N_0^K$.
We next show that there exists a set $V' \subset V$ such that V' covers E and $|V'| \leq K$ if and only if Q is M-bounded under $\mathcal{A}$.

$\boxed{\Rightarrow}$ When there exists a set $V' = \{v_{k_j} \mid j \in [1, K]\}$ and $k_j \in [1, |V|]$ such that V' covers E of G. Let X_C' be the subset of X_C that consists of all attributes that correspond to nodes in V'. By the definition of $\mathcal{A}_0$ and Lemma 13(2), we have $X_C \mapsto_{\mathcal{I}_E^M} (\mathcal{U}_{k_j} \cup X_C, N_0)$ for each $j \in [1, K]$. Since V' covers E, we have $\bigcup_{j=1}^K \mathcal{U}_{k_j} = \bigcup_{i=1}^{|E|} U_i$. Thus, $X_C \mapsto_{\mathcal{I}_E^M} (R, N_0^K)$. Note that X_C is indexed, by Lemma 13(2), we know Q is effectively M-bounded.

$\boxed{\Leftarrow}$ When Q is effectively M-bounded, by Lemma 13(2), we know that $X_C \mapsto_{\mathcal{I}_E^M} (\bigcup_{i=1}^{|E|} U_i, N_0^K)$. By the definition of $\mathcal{A}$, there must exist a subset X_V that consists of K attributes $V_{k_1}, \ldots, V_{k_K}$ of R ($k_j \in [1, n]$, for all $j \in [1, K]$), such that $X_V \mapsto_{\mathcal{I}_E^M} (\mathcal{U}_{k_j}, N_0)$ for all $j \in [1, K]$. Let V' be the set consisting of all nodes in V of G that are encoded by X_V. By the definition of $\mathcal{A}$, V' covers E of G with K nodes.

Upper bound. The NP upper bound is verified by the following NP algorithm: first guess a proof with $\mathcal{I}_E^M$ of length bounded by $O(|Q|^2|\mathcal{A}|^3)$, and then verify the proof, and check (a) whether it leads to $X_C \mapsto_{\mathcal{I}_E^M} (\bigcup_{i=1}^n X_Q^i, M)$ and (b) whether $\bigcup_{i=1}^n X_Q^i$ is indexed in $\mathcal{A}$. Note that verification and checking can be done in PTIME, thus it is an NP algorithm. That is, $\mathsf{EBnd}(Q, \mathcal{A}, M)$ is in NP.

We next study DP and MDP in the case when M is part of the input, with rules in $\mathcal{I}_E^M$ and Lemma 13(2) and (3).

Corollary 14: *When M is part of the input,*
(1) $\mathsf{DP}(Q, \mathcal{A}, M)$ *is* NP*-complete; and*
(2) $\mathsf{MDP}(Q, \mathcal{A}, M)$ *is* NPO*-complete.* □

Proof: (1) The NP-hardness lower bound is obviously since $\mathsf{DP}(Q, \mathcal{A}, M)$ contains $\mathsf{DP}(Q, \mathcal{A})$ as a special case where M is sufficiently large. The upper bound can be verified the following NP algorithm: first guess a set X_P of nontrivial parameters and a proof with length bounded by $O(|Q|^2|\mathcal{A}|^3)$, and then verify the proof and check (a) whether it leads to $X_P \cup X_C \mapsto_{\mathcal{I}} (\bigcup_{i=1}^n X_Q^i, M)$, and (b) whether $\bigcup_{i=1}^n X_Q^i$ is indexed in $\mathcal{A}$. Note that the verification and checking is in PTIME and thus the problem is in NP.
(2) The NPO-completeness follows from (a) $\mathsf{DP}(Q, \mathcal{A}, M)$ is in NP and (b) $\mathsf{DP}(Q, \mathcal{A})$ is NPO-hard, which is a special case

of $\mathsf{DP}(Q, \mathcal{A}, M)$ with sufficiently large M. □

Appendix C: Algorithms

Query plan from proofs with rules in $\mathcal{I}_E$

Suppose that $X_C \mapsto_{\mathcal{I}_E} (X_Q^i, M_i)$ is proven by $\rho_i = \varphi_1, \cdots, \varphi_m$, where φ_j denotes application of a rule in $\mathcal{I}_E$. We show that given D, ρ_i tells us how to find a list of subsets $T_1, \ldots, T_m$ of D such that

- $D_Q^i = \bigcup_{j=1}^{m} T_j$ and $D_Q = \bigcup_{i=1}^{n} D_Q^i$, and
- for all $j \in [1, m]$, $T_j \subseteq D$, T_j has at most N_j tuples and can be fetched by using indices in $\mathcal{A}$, where N_j is a number deduced from the proof, independent of $|D|$.

We can then compute $Q(D)$ by conducing joins and projections on these T_j's only, guided by conditions in σ_C of Q, as illustrated by how we get $Q_0(D_0)$ using T_1–T_4 in Example 1.

Below we show how to fetch T_j from D guided by rule φ_j. Initially, $T_1 = \bigcup_{j=1}^{n} \sigma_{X_j = C_j}(D)$, and can be fetched by using indices in $\mathcal{A}$ on the constants of X_C (see Theorem 4 and its proof).

(a) When φ_j is Reflexivity, we get T_j by projecting from $T_{j'}$ on X', where $j' < j$ and $T_{j'}$ is fetched in previous steps in the proof, without accessing to original data.

(b) When φ_j actualizes a constraint $X \to (Y, N)$ of $\mathcal{A}$, we fetch N tuples for T_j either from D by using index in $\mathcal{A}$ on X for Y, or from a bounded subset $T_{j'}$ of D ($j' < j$) deduced from previous steps in proof ρ_i, on which φ_j is applied.

(c) When φ_j is Transitivity, we fetch T_j by firstly fetching a set S_1^X with accessing no more than N tuples (*i.e.,* $|S_1^X| \leq N$), and then for each tuple in S_1^X, we fetch $S_2^{X_1}$ with accessing no more than N' tuples from D. Thus, the total number of tuples required to access is bounded by $N * N'$. Note that, these $N * N'$ tuples may not be all valid in the current data D, but we will handle this with Combination, since by Theorem 4, there must be a step $\varphi_{j'}$, where $j' > j$, such that $\varphi_{j'}$ is a combination rule that combines W with X.

(d) When φ_j is Combination, we get T_j as follows. Denote $\bigcup_{s=1}^{j-1} T_s$ by T. As indicated by the rule (Fig. 2), for $l \in [1, k]$, (i) all X_l and Y_l values are already fetched in T; and (ii) we can check whether these X_l and Y_l values appear in tuples of D, *i.e.,* they are contained in the projection of D on $\bigcup_{l=1}^{k} X_l \cup Y_l'$, by using the indices on the attributes. There are at most $N_1 * \ldots * N_k$ such tuples from T to be inspected in D, and T_j consists of these tuples.

Note that Augmentation is a special case of Combination with $k = 1$. Thus the interpretation of Augmentation is the same to the Combination.