Bounded Evaluation: Querying Big Data with Bounded Resources

Citation for published version:
Cao, Y, Fan, W & Yuan, T 2020, 'Bounded Evaluation: Querying Big Data with Bounded Resources', International Journal of Automation and Computing, vol. 17, no. 4, pp. 502–526. https://doi.org/10.1007/s11633-020-1236-1

Digital Object Identifier (DOI):
10.1007/s11633-020-1236-1

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
International Journal of Automation and Computing

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Bounded Evaluation: Querying Big Data
with Bounded Resources

Yang Cao¹  Wen-Fei Fan¹,²,³  Teng-Fei Yuan¹

¹University of Edinburgh, Edinburgh EH8 9AB, UK
²Shenzhen Institute of Computing Sciences, Shenzhen University, Shenzhen 518060, China
³Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University, Beijing 100191, China

Abstract: This work aims to reduce queries on big data to computations on small data, and hence make querying big data possible under bounded resources. A query \( Q \) is boundedly evaluable when posed on any big dataset \( D \), there exists a fraction \( D_Q \) of \( D \) such that \( Q(D) = Q(D_Q) \), and the cost of identifying \( D_Q \) is independent of the size of \( D \). It has been shown that with an auxiliary structure known as access schema, many queries in relational algebra (RA) are boundedly evaluable under the set semantics of RA. This paper extends the theory of bounded evaluation to RAagg, i.e., RA extended with aggregation, under the bag semantics. (1) We extend access schema to bag access schema, to help us identify \( D_Q \) for RAagg queries \( Q \). (2) While it is undecidable to determine whether an RAagg query is boundedly evaluable under a bag access schema, we identify special cases that are decidable and practical. (3) In addition, we develop an effective syntax for boundedly evaluable RAagg queries, i.e., a core subclass of boundedly evaluable RAagg queries without sacrificing their expressive power. (4) Based on the effective syntax, we provide efficient algorithms to check the bounded evaluability of RAagg queries and to generate query plans for boundedly evaluable RAagg queries. (5) As proof of concept, we extend PostgreSQL to support bounded evaluation. We experimentally verify that the extended system improves performance by orders of magnitude.

Keywords: Bounded evaluation, resource-bound query processing, effective syntax, access schema, boundedness.

1 Introduction

Querying big data can be prohibitively costly. As an indicator, it is NP-hard¹ to decide whether a tuple is in the answer \( Q(D) \) in a dataset \( D \) to an SPC (select, project, Cartesian product) query \( Q \), and it is PSPACE-hard² when \( Q \) is a query in relational algebra (denoted by RA)³. It takes days to join two tables with 10 million tuples each⁴. One might be tempted to think that parallel computation could do the job. However, there exist computational problems for which parallel scalability is beyond reach, i.e., no matter how many machines are used, the parallel runtime of algorithms for such problems may not be reduced⁵. Worse still, small businesses typically have constrained resources and may not afford large-scale parallel computation.

Is querying big data beyond the reach of small companies, or is it just a privilege of big companies? Is it possible to extend DBMS with an immediate capacity to answer common queries over big datasets under constrained resources?

One approach to tackling the challenge has recently been studied, based on bounded evaluation⁶, ⁷. To answer a query \( Q \) on a dataset \( D \), the idea is to look at only a ‘bounded’ fraction \( D_Q \) of \( D \) that suffices to compute \( Q(D) \), instead of at the entire \( D \). This is doable by using an access schema \( A \), which is a combination of cardinality constraints and associated indices. Under \( A \), \( Q \) is boundedly evaluable if for all datasets \( D \) that conform to \( A \), one can identify \( D_Q \subseteq D \) by reasoning about the cardinality constraints, and fetch \( D_Q \) by using the indices of \( A \), such that (a) \( Q(D_Q) = Q(D) \) and (b) \( D_Q \) is determined by \( A \) and \( Q \) only. In other words, if \( Q \) is boundedly evaluable under \( A \), query \( Q \) can then be exactly answered via bounded evaluation, by accessing only \( D_Q \) of size bounded by the cardinalities in \( A \).

The theory has been tested in industry and is found to “improve the performance by orders of magnitude”⁷.

Example 1. Consider query \( Q_1 \) from Facebook Graph Search⁸: Find all my friends who have check-ins in UK². The query is posed on dataset \( D_t \) with two relations: (a) friend (uid, fid), stating that person fid is a friend of uid, and (b) checkin (uid, loc, cty, date), stating that person uid checked in at location loc in country cty on date. Written as an RA query, \( Q_1 \) is as follows (\( u_q \) denotes “me”):

³See [1] for more about complexity classes, e.g., NP and PSPACE.
⁴Facebook users can check-in to locations via “check-ins.”
$Q_1(x) = \text{friend}(u_0, x) \land \text{checkin}(x, \text{loc}, \text{"UK"}, \text{date})$.

Here dataset $D_1$ is big, with trillions of friend links and check-ins$[5]$. It is costly to compute $Q_1(D_1)$ directly.

Now consider a set $A_1$ of real-life cardinality constraints:

\[ \phi_1: \text{friend}(\text{pid} \rightarrow \text{fid}, 5000); \]
\[ \phi_2: \text{checkin}(\text{uid} \rightarrow \text{country}, 193). \]

Here constraint $\phi_1$ specifies a Facebook policy$[10]$: a limit of 5000 friends per user; and $\phi_2$ states that each user can check-in at most 193 countries. Indices can be built on $D_1$ based on $\phi_1$ such that given a person, it returns the ids of all her friends by accessing at most 5000 friend tuples; similarly for $\phi_2$. Taken together, these constraints and their associated indices are called access constraints$[5]$.

Using $A_1$, we can compute $Q_1(D_1)$ by accessing at most 970000 tuples from $D_1$, instead of trillions. (1) We fetch $T_1$ of at most 5000 fid's of friend tuples with uid = $u_0$, by using $\phi_1$. (2) For each fid f in $T_1$, we fetch $T_2$ of at most 193 country values with $\phi_2$. (3) We return the set of fid's in $T_1$ with country = UK in $T_2$. The plan fetches at most 5000 + $193 \times 5000$ tuples to compute $Q_1(D_1)$, no matter how big $D_1$ is. Hence, $Q_1$ is boundedly evaluable under $A_1$.

As shown in Example 1, bounded evaluation answers a query $Q$ over a big dataset by accessing a set $D_Q$ of data values with bounded size. It does this by retrieving values (i.e., partial tuples) using indices associated with cardinality constraints that correlate attributes. One might think that this can also be carried out by conventional index-only plans for query optimization$[11]$. However, the two are different problems as indicated by their complexity bounds: deciding whether an SPC query can be answered with a “bounded” query plan is EXPTIME-hard$[5]$, while it is in PTIME to decide whether it has an index-only plan$[11]$.

While bounded evaluation is promising, more work has to be done, from theory to systems. Bounded evaluation has only been studied for RA queries under the set semantics$[5, 6]$. In the real world, queries are often expressed in RA$_{agg}$, i.e., RA extended with aggregation under the bag semantics. RA$_{agg}$ can express all SQL (structured query language) queries that do not carry arithmetic expressions. This makes bounded evaluation more intriguing.

**Example 2.** Recall query $Q_1$ and access schema $A_1$ from Example 1. Consider query $Q_2$ to find the number of UK check-ins from each of my friends. Written in RA$_{agg}$,

\[ Q_2 = \text{gpBy}(Q_3, \text{uid}, \text{count}(\text{cty})), \text{ where } \]
\[ Q_3 = \pi_{\text{uid}, \text{cty}}(\text{friend}(u_0, x) \land \text{checkin}(x, \text{loc}, \text{"UK"}, \text{date})). \]

Here gpBy($Q_3$, uid, count(cty)) groups the results of $Q_3$ by attribute uid and calculates count(cty) for each group (see Section 2.1 for more details about gpBy operator). In contrast to $Q_1$, $A_1$ does not help us answer $Q_2$.

Using $\varphi_2$, we can fetch a set of distinct countries for each friend $x$. However, $x$ may have multiple UK check-ins. Access schema no longer suffices for RA$_{agg}$ under the bag semantics.

For practical use to emerge from the study, it is necessary to extend bounded evaluation from RA to RA$_{agg}$ (SQL). This gives rise to several questions. How should we extend the access schema of $[5, 6]$ to support the bag semantics? We will see that the problem for checking whether an SQL query is boundedly evaluable is undecidable. Given the negative result, is bounded evaluation beyond reach in practice? More specifically, is there any practical and decidable special case? Is it possible to develop a systematic method that allows us to efficiently check the bounded evaluable of SQL queries? In addition, after determining that an SQL query is boundedly evaluable, how can we generate and optimize a query plan to carry out its bounded evaluation?

**Contributions.** This paper answers these questions by extending the study to RA$_{agg}$, from theory to practice.

(1) **Bounded evaluation for SQL.** We extend bounded evaluation from RA to RA$_{agg}$, i.e., SQL (without arithmetic) to support arbitrarily nested aggregate sub-queries and group-by clauses. We introduce bag access schemas, an extension of the access schema of $[5, 6]$ to support the bag semantics. We also formulate bounded query plans for RA$_{agg}$.

(2) **Complexity of bounded evaluation.** Not surprisingly, bounded evaluable is undecidable for SQL since it is already undecidable for RA$[5]$. We identify practical conditions that cover a number of real-life queries, for which the bounded evaluable can be efficiently determined. These conditions tell us what makes queries bounded.

(3) **Effective syntax.** To accommodate the undecidability, we develop an effective syntax $\mathcal{L}_B$ for boundedly evaluable RA$_{agg}$ queries. We show that under a bag access schema $B$, (a) an RA$_{agg}$ query $Q$ is boundedly evaluable if and only if it is equivalent to a query $Q' \in \mathcal{L}_B$; and (b) it is in PTIME (polynomial time) to check whether $Q$ is in $\mathcal{L}_B$. That is, $\mathcal{L}_B$ is a core subclass of boundedly evaluable RA$_{agg}$ queries that are syntactically checkable without sacrificing the expressive power. This is along the same lines as how commercial database systems (DBMS) deal with safe relational calculus queries, which are undecidable to decide$[12, 14]$.

(4) **Extending DBMS with bounded evaluation.** We present a framework, referred to as BEAS (bounded evaluable SQL) to provide commercial DBMS with the capability of bounded evaluation of RA$_{agg}$ queries. Given an RA$_{agg}$ query $Q$ and a bag access schema $B$, BEAS first checks whether $Q$ is boundedly evaluable under $B$. If so, it generates a query plan for $Q$ to compute $Q(D_B)$ by accessing a bounded small fraction $D_Q$ of $D_B$ using $B$. Otherwise, it leverages access schema and generates a partially bounded plan, to bound sub-queries of $Q$. We develop al-

© Springer
(5) Experimental study. As proof of concept, we extend PostgreSQL with bounded evaluation, denoted by BEAS@PG. Using TPCH benchmark[15] and real-life datasets, we evaluate the performance of BEAS@PG compared to PostgreSQL. We find that BEAS@PG improves PostgreSQL by up to 4 orders of magnitude for boundedly evaluable queries.

Querying big data under bounded resources.

This work is a component of a framework for querying big data. As outlined in [16], the framework works as follows: given an SQL query \( Q \) posed on a big dataset \( \mathcal{D} \),

1. if it first checks whether \( Q \) is boundedly evaluable;
2. if so, it computes the exact answer \( Q(\mathcal{D}) \) by accessing a bounded fraction \( D_Q \) of \( \mathcal{D} \) via bounded evaluation;
3. otherwise, it computes approximate answers to \( Q \) in \( \mathcal{D} \) also by accessing a bounded amount of data, and provides deterministic accuracy ratios[16].

In the entire process, it only accesses a bounded fraction of data and can be conducted under bounded resources. Hence it is feasible to provide small businesses with a capacity of querying big data despite constrained resources.

To simplify the discussion, we focus on row-oriented DBMS (a.k.a. row stores) in this paper. Nonetheless, as will be seen in Section 2, the model of bounded evaluation subsumes column stores. Moreover, bounded evaluation can be readily extended to parallel and distributed systems[18].

Organization. The remainder of the paper is organized as follows. Section 2 defines bag access constraints and formulates boundedly evaluable RA\textsubscript{agg} queries. Section 3 studies the complexity of bounded evaluation for RA\textsubscript{agg} queries. Section 4 proposes effective syntax \( L_B \) for boundedly evaluable RA\textsubscript{agg} queries. Section 5 introduces BEAS and develops its underlying algorithms. The experimental study is presented in Section 6. We discuss related work in Section 7, and identifies topics for future work in Section 8.

2 Bounded evaluation of SQL queries

We first define bag access schema (Section 2.1) and then formulate bounded evaluation of RA\textsubscript{agg} queries, aggregate or not, under the bag semantics (Section 2.2).

2.1 RA\textsubscript{agg} and bag access schema

We start with a review of RA\textsubscript{agg}, an extension of RA with a group-by construct and nested aggregate sub-queries.

RA\textsubscript{agg} queries. An RA\textsubscript{agg} query is a query defined in terms of RA operators (i.e., select \( \sigma_C \), project \( \pi_Y \), Cartesian-product \( \times \) or join \( \bowtie_C \), renaming \( \rho_{A \rightarrow B} \), union \( \cup \) and set difference \( - \)), and additionally a group-by aggregate operator

\[
\text{gpBy}(Q, X, \text{agg}_1(V_1), \cdots, \text{agg}_m(V_m))
\]

where (a) \( Q \) is an RA\textsubscript{agg} query, (b) \( X \) is a set of attributes for group-by, (c) \text{agg} is one of aggregate functions max, min, count, sum, avg, and (d) \( V_1, \cdots, V_m \) are attributes such that \( X \cup \bigcup^n_{i=1} \{\text{agg}_i(V_i)\} \) forms the output relation of \( Q \). We refer to \text{agg}(V_i) as an aggregate field on attribute \( V_i \). We write the operator as \( \text{gpBy}(Q, X, \text{agg}(V)) \) when it is clear from the context. Since \( Q \) may include aggregate operators itself, aggregations in an RA\textsubscript{agg} query may be arbitrarily nested.

In SQL syntax, the operator can be written as

\[
\begin{align*}
\text{select} & \quad \text{agg}_1(V_1), \cdots, \text{agg}_m(V_m) \\
\text{from} & \quad Q \\
\text{group by} & \quad X
\end{align*}
\]

As a special case, when \( X = \emptyset \), \( \text{gpBy}(Q, X, \text{agg}(V)) \) does not have a group-by construct. We write it simply as \( \text{agg}(Q) \).

Example 3. Query \( Q_2 \) in Example 2 is an RA\textsubscript{agg} query.

As another example, an RA\textsubscript{agg} query with nested aggregation is \( Q_4 \) over relations \( R(A, B, C) \) and \( S(E, F, W) \):

\[
Q_4 = \text{avg}(\sigma_A(Q_3(y) \bowtie S(y, 1, z))),
\]

where

\[
Q_3: \quad y = \sum(\sigma_F(R(w, 1, u) \bowtie R(w, x, y)));
\]

Bag access schema. To support the bag semantics, we extend the access schema of [5, 19] to bag access schema. Over a database schema \( \mathcal{R} \), a bag access schema \( B \) is a set of bag access constraints of the form:

\[
\varphi = R[\{X \rightarrow Y, N\}]
\]

where \( R \) is a relation schema in \( \mathcal{R} \), \( X \) and \( Y \) are sets of attributes of \( R \), and \( N \) is a positive integer.

To define the semantics of bag access constraints, we use the following notations. (1) Denote by \( \mathcal{D} \) a database of \( \mathcal{R} \), and by \( D \) an instance of relation schema \( \mathcal{R} \) in \( \mathcal{D} \).

2. For an instance \( D \) of \( R \), \( D_Y(X = \bar{a}) = \{[Y] \mid t \in D, t[X] = \bar{a}\} \), i.e., \( D_Y(X = \bar{a}) \) denotes the set of \( Y \) values corresponding to \( X\)-value \( \bar{a} \). (3) For any \( XY\)-value \( \overline{ab} \) in \( D \), \( m(D, \overline{ab}) \) denotes the cardinality of the bag (multiset) \( \{t \in D \mid t[XY] = \overline{ab}\} \), i.e., the number of occurrences of \( \overline{ab} \) as \( XY \) attributes in \( D \); we refer to \( m(D, \overline{ab}) \) as the multiplicity of \( \overline{ab} \) in \( D \).

We say that \( D \) conforms to \( \varphi \), denoted by \( D \models \varphi \), if

1. for any \( X\)-value \( \bar{a} \) in \( D \), \( m(D, \overline{ab}) \leq N \), i.e., there exist at most \( N \) distinct associated \( Y\)-values in \( D \); and
2. there exists an index for \( \varphi \) on \( D \) such that given any \( X\)-value \( \bar{a} \), by accessing at most \( N \) tuples, it retrieves (a) all associated distinct \( Y\)-values \( \bar{b} \) in \( D \), and (b) for each such \( \bar{b} \), the multiplicity \( m(D, \overline{ab}) \).

Intuitively, \( D \models \varphi \) if for any \( X\)-value, there exist at
most $N$ distinct corresponding $Y$ values in $D$. Moreover, these $Y$-values (partial tuples) and their multiplicities in $D$ are indexed by $\psi$ and can be efficiently retrieved via the index.

**Example 4.** Extending $A_1$ of Example 1, a bag access schema $B_1$ consists of the following bag access constraints:

- $\varphi_1 = \text{friend}(\text{pid} \to \text{fid}, 5000);$
- $\varphi_2 = \text{checkin}(\text{uid} \to \text{country}, 193).$

Here $\varphi_2$ says that (a) for any uid $u_1$, there exist at most 193 distinct country values, and (b) there exists an index built on the friend relation that given any uid $u_1$, fetches all associated distinct countrys $c$ and for each country $c$, the multiplicity of $(u_1, c)$ in the friend relation for country $c$; similarly for $\varphi_2$.

As another example, consider a bag access schema $B_2$ for $Q_4$ of Example 3, which consists of bag access constraints:

- $\varphi_3 = R([A \to B, 1]);$
- $\varphi_4 = R([B \to C, 10]);$
- $\varphi_5 = S([EF \to W, 10]).$

We will see that $Q_4$ can be efficiently answered with $B_2$.

A database instance $D$ of $R$ conforms to a bag access schema $B$, denoted by $D \models B$, if $D \models \varphi$ for every $\varphi \in B$, where $\varphi = R([X \to Y, N])$, and $D$ is the instance of $R$ in $D$.

Intuitively, a bag access constraint $\varphi$ extends an access constraint $\psi$ of [5, 19] by incorporating multiplicity. Similar to $\psi$, given any $X$-value $\overline{a}$ in $D$, $\varphi$ enforces the cardinality constraint $|D_X(\{X = \overline{a}\})| \leq N$ and returns distinct corresponding $Y$-values. In contrast to $\psi$, for each corresponding $Y$-value $\overline{b}$, $\varphi$ also returns the multiplicity of $\overline{ab}$ in $D$. In other words, access constraints under the set semantics\cite{6} are a special case of bag access constraints when we only bound the cardinality and retrieve distinct values.

**Remark 1.** When it is clear from the context, we also simply refer to bag access schema as access schema.

### 2.2 Bounded evaluation of RA$_{aggr}$ queries

We next define bounded evaluation for RA$_{aggr}$ queries.

**Multiplicity relations.** From an instance $D$ of a relation schema $R$, we can use the index of a bag access constraint $R([X \to Y, N])$ to retrieve a relation consisting of tuples $(t([X, Y]), m([X, Y]))$, where $t$ is a tuple in $D$. It is a set that beside partial tuples $t([X, Y])$ in $D$, carries multiplicity $m([D, t([X, Y]])$, and is referred to as a multiplicity relation.

The RA$_{aggr}$ operators can be readily extended to multiplicity relations. Take join operator $\Join_C$ as an example. Given two multiplicity relations $I_1$ and $I_2$, the result of $I_1 \Join_C I_2$, denoted by $I_0$, is a multiplicity relation as follows: (a) tuples in $I_0$ have the form $(t, M)$, where $t$ is a result tuple of $I_1 \Join_C I_2$ using the conventional join semantics (ignoring multiplicity), and (b) $M = m(I_1, t_1) \times m(I_2, t_2)$, where $t$ is joined from $t_1 \in I_1$ and $t_2 \in I_2$, and $m(I_i, t_i)$ denotes the multiplicity of tuple $t_i$ in multiplicity relation $I_i(i \in \{1, 2\})$. Similarly, other RA$_{aggr}$ operators are defined on multiplicity relations.

**Bounded RA$_{aggr}$ plans.** A bounded RA$_{aggr}$ plan $\xi$ under a bag access schema $B$ is an algebra tree that extends conventional RA$_{aggr}$ query plans with a new operator $\text{fetch}(T, \varphi)$, where $\varphi = R([X \to Y, N])$ is a bag access constraint in $B$, and $T$ is an intermediate multiplicity relation on attributes $R[X]$ (see Appendix for a formal definition). Over an instance $D$ of $R$ that conforms to $\varphi$, fetch $(T, \varphi)$ retrieves an intermediate relation $S = \bigcup_{a \in T_X} D_X(\{X = a\})$ by using the index of $\varphi$ on $D$, where each tuple in $S$ is annotated with multiplicity $m(D, t)$ (also retrieved by $\varphi$).

Intuitively, a bounded RA$_{aggr}$ plan starts with a set of constants (possibly $\emptyset$), retrieves data from $D$ via the fetch operator, and applies RA$_{aggr}$ operators to the fetched data, except that it accesses data by employing the indices of the bag constraints in $B$ only, and allows group-by aggregate and operates on multiplicity relations.

We denote by $(\xi(D))$ the result of applying plan $\xi$ to $D$.

**Example 5.** Recall RA$_{aggr}$ query $Q_2$ from Example 2 and bag access schema $B_1$ from Example 4. A bounded plan $\xi Q_2$ for $Q_2$ under $B_1$, written in algebra expressions, is as follows:

$T_1 = \text{fetch}(\text{uid, fid});$
$T_2 = \text{fetch}(\text{uid, cty});$
$T_3 = \sigma_{\text{cty}}(T_2);$
$T_4 = \text{gpBy}(T_3, \text{uid, count(cty)}).$

As a more intriguing example, recall query $Q_4$ from Example 3 and bag access schema $B_2$ from Example 4. By the cardinality constraint in $\varphi_2$, for each $A$-value in any database instance $D \models B_2$ for $w$ of $Q_4$, there exists at most 1 distinct $B$-value associated with $w$. Therefore, $Q_4 \equiv_{B_2} Q_6$, i.e., $Q_4$ is equivalent to $Q_6$ on every database $D \models B_2$, where $Q_6$ is

$Q_6 = \text{avg}(\sigma_{\text{cty}}(Q_4(y) \Join \text{sum}(S(y, 1, z))));$

Under $B_2$, $Q_6$ (hence $Q_4$) has a bounded query plan $\xi Q_4$:

$T_1(B, C) = \text{fetch}([1], \varphi_4);$
$T_2 = \text{sum}(\sigma_{\text{cty}}(T_1));$
$T_3 = \text{fetch}(T_2 \times [1], \varphi_5);$
$T_4 = \text{avg}(\pi_W(T_3)).$

Observe that $\xi Q_4$ does not explicitly use $\varphi_3$. However, the correctness of $\xi Q_4$ relies on the cardinality of $\varphi_3$. Moreover, $\xi Q_4$ propagates constants of $Q_4$ via join and fetch, such that all values and their combinations that are needed for answering $Q_4$ are fetched from $D \models B_2$. In particular, in the presence of nested aggregation, answers to aggregate sub-queries can also be used by fetch, e.g., $T_3$.

**Boundedly evaluable queries.** Under access schema $B$, an RA$_{aggr}$ $Q$ is boundedly evaluable if it has a
plan $\xi$ such that:
- $\xi$ is a bounded RA$_{agg}$ plan under $B$;
- constants $[c]$ in $\xi$ are from selection conditions of $Q$;
- moreover, for any database $D \models B$, $\xi(D) = Q(D)$.

We write $Q \equiv_{\equiv_{\xi}} Q$ if $Q$ has such a bounded query plan $\xi$.

Here for any multiplicity relation $D_1$ and a conventional bag (multiset) $D_2$, we write $D_1 = D_2$ if $D_2$ can be obtained from $D_1$ by including $m$ copies of each tuple $(t, m) \in D_1$.

For example, query $Q_2$ of Example 2 is boundedly evaluable under $B_1$ of Example 4, since it has a bounded plan $\xi_{Q_2}$ given in Example 5. Similarly, $Q_4$ of Example 3 is also boundedly evaluable under $B_2$ of Example 4.

Observe the following about bounded RA$_{agg}$ plans $\xi$.

1. **Scale independence.** Each fetch operation in $\xi$ retrieves data with a cost that can be quantified by the bag constraint employed. Hence the cost of executing $\xi$ is determined by bag access schema $B$ and query plan $\xi$ only, not by the size of dataset $D$ as long as $D$ conforms to $B$.

That is, under the bag semantics, bounded RA$_{agg}$ plans preserve the scale independence of bounded evaluation for RA queries$^{[5, 6]}$.

2. **Late bag semantics enforcement.** Plan $\xi$ fetches and operates on sets since fetch($T, \psi$) returns a set. It defers the process of the bag semantics to a stage as late as possible. This reduces performance degradation caused by duplicated values in, e.g., joins, in which duplicates get inflated rapidly.

3. **Subsuming column-stores.** Bound plan $\xi$ can also express query evaluation over column-stores$^{[20]}$ or column-store indices$^{[21]}$. Indeed, in a column store (or a column-store index), each column (or column index) on attribute $A$ of a relation schema $R$ is essentially a special case of bag access constraint of the form $R[\emptyset \rightarrow A, N]$. Hence, column store and columnstore index are a special case of bag access schema and hence their evaluation plan can be expressed by a bound plan $\xi$ under such a bag access schema.

Note that the efficiency of column stores mainly comes from its implementation-level optimization, e.g., column compression and vectorization$^{[20]}$. While these optimization strategies can also be used to implement the indices of bag access schema, these are not the focus of this paper. We study query evaluation at the logical level, under generic constraints $R[X \rightarrow Y, N]$ when $X$ is not necessarily empty. Hence, this paper focuses on row-oriented databases as the underlying platform for implementing bag access schema.

3 **Complexity of bounded evaluation**

In this section, we study the complexity of bounded evaluability and identify practical decidable cases.

**Bounded evaluability.** The problem is stated as follows.

- **Input:** A database schema $R$, a bag access schema $B$ over $R$, and an RA$_{agg}$ query $Q$ over $R$.
- **Question:** Is $Q$ boundedly evaluable under $B$?

This bounded evaluability problem is to decide whether a query can be answered by accessing a bounded amount of data, and is underlying the first step of our framework for querying big data under bounded resources (Section 1).

No matter how important, the problem is hard. To see why it is intriguing, let us consider Example 6.

**Example 6.** Consider bag access schema $B_3$ and SPC query $Q_5$ defined on relations $T(A, B)$ and $U(E, F)$:

- $\xi_{B_3}$ consists of the following two access constraints:
  - $\varphi_6 = T(A \rightarrow B, N)$;
  - $\varphi_7 = U(E \rightarrow F, 2)$.
- query $Q_5$ is $Q_5 = Q_0 - Q_{10}$, where
  - $Q_0 = \pi_y(T(x, y) \Join U(w, 1) \Join U(w, x) \Join U(w, y))$, and
  - $Q_{10} = \pi_z(T(z, z) \Join U(u, 1) \Join U(u, z))$.

At a first glance, none of $Q_0$ and $Q_{10}$ seems boundedly evaluable, and hence neither is $Q_5$. Indeed, we cannot retrieve values for any of $x$, $y$, or $z$ using indices in $B_3$.

However, putting together $U(w, 1) \Join U(w, x) \Join U(w, y)$ and $\varphi_7$ of $B_3$, one can deduce that $x$ must be equal to either $1$ or $y$ in all tuples retrieved from instance of $T$ by any query plan for $Q_0$. In other words, under $B_3$, $Q_0$ reduces to $SPCU. Q_0 = Q_5 \cup Q_{10}$, where $Q_5 = \pi_y(T(1, y) \Join U(w, 1) \Join U(w, , y) \Join U(w, y))$, and $Q_{10} = \pi_z(T(z, z) \Join U(u, 1) \Join U(u, z))$.

Hence, $Q_5 = (Q_0 \cup Q_{10}) - Q_{10} = Q_5$ since $Q_5 \equiv Q_{10}$. It is easy to see that $Q_5$ is boundedly evaluable under $B_3$ and as a result, so is $Q_5$.

As shown above, it is often necessary to check query equivalence to decide whether a query is bound. The use of union ($\cup$) allows us to convert SPC to SPCU under a bag access schema (e.g., $Q_1$ to $Q_3 \cup Q_7$), which may further interact with set difference (−) (e.g., $Q_1$ and $Q_7$).

It is beyond reach in practice to check the equivalence of RA or RA$_{agg}$ queries. Thus, the bounded evaluability problem is already undecidable for RA, a special case of RA$_{agg}$.

**Theorem 1.**$^{[5]}$ The bounded evaluability problem is undecidable for RA queries.

Theorem 1 was verified under access schema. As remarked in Section 2, access schema is a special case of bag access schema. Hence, the bounded evaluability problem remains undecidable for RA under a bag access schema. As an immediate corollary, the bounded evaluability problem is undecidable for RA$_{agg}$, which subsumes RA.

**Decidable cases.** We next identify special cases, when the bounded evaluability is decidable. The reason is twofold. (1) The special cases cover a large number of RA$_{agg}$ queries used in practice, e.g., all SPC sub-queries of built-in benchmark queries in TPCH$^{[15]}$ and TPCDS$^{[22]}$.

(2) These cases reveal insight about why queries become boundedly evaluable. In Section 4, we will deal with gen-
eric RA\textsubscript{agg} queries, by devising an effective syntax for boundedly evaluable RA\textsubscript{agg} queries, sub-queries.

(1) PTIME cases. The first special case, denoted by \mathcal{C}^{P}, consists of combinations of SPC queries and bag access schema for which the bounded evaluability can be checked in PTIME, covering all SPC sub-queries of TPCH and TPDCS.

Class \mathcal{C}^{P}. For any bag access schema \(B\) and SPC query \(Q\) over the same database schema \(R,(B,Q)\in \mathcal{C}^{P}\) if
(a) for each bag constraint \(\varphi = R(X	o Y,N)\), \(N \geq |Q|\), where \(|Q|\) is the number of relation atoms in \(Q\); and
(b) \(Q\) has no self-join.

Theorem 2. For any bag access schema \(B\) and SPC \(Q\),
(1) it is in PTIME to decide whether \((B,Q)\) is in \(\mathcal{C}^{P}\); and
(2) it is in PTIME to decide whether \(Q\) is boundedly evaluable under \(B\) if \((B,Q)\) is in \(\mathcal{C}^{P}\).

Proof. Statement (1) apparently holds. Below we prove (2) by giving a PTIME sufficient and necessary condition for checking the bounded evaluability of \(Q\) under \(B\).

The condition needs a notion of covered SPC queries from [6]. It is characterized by a set \(\text{cov}(Q,B)\), which consists of attributes whose values can be retrieved via fetch operations along with \(B\), without directly accessing raw data in a database. More specifically, let \(X^{Q}_{B}\) be the set of attributes \(A\) in the constant selection predicates of \(Q\), i.e., \(\sigma_{A=c}\), for a constant \(c\). Then \(\text{cov}(Q,B)\) is inductively defined as:

(a) \(X^{Q}_{B} \subseteq \text{cov}(Q,B)\); 
(b) if \(A \in \text{cov}(Q,B)\) and \(\Sigma_{Q} \vdash A = B\) (denoting that \(A = B\) can be deduced from selection predicates \(\Sigma_{Q}\) via the transitivity of equality), then \(B \in \text{cov}(Q,B)\);
(c) if \(X \subseteq \text{cov}(Q,B)\) and \(R(X	o Y,N)\) \in \(B\), then \(Y \subseteq \text{cov}(Q,B)\); and
(d) \(\text{cov}(Q,B)\) contains nothing else.

Denote by \(X^{Q}_{B}\) the set of attributes that are either in the selection or join predicates of \(Q\), or are the top-level projection attributes. Then we show the following.

Lemma 3. For any \((B,Q)\in \mathcal{C}^{P}\), \(Q\) is boundedly evaluable under \(B\) if and only if \(\varphi = R(X	o Y,N)\) \in \(B\) such that \(X^{Q}_{R} \subseteq XY \subseteq \text{cov}(Q,B)\).

From Lemma 3, Theorem 2(2) follows since one can simply check the condition of Lemma 3 in PTIME in the sizes of \(Q\) and \(B\). Below we prove Lemma 3.

(\(\Rightarrow\)) Assume that \(Q\) is boundedly evaluable under \(B\).

Then there exists a bounded plan \(\xi\) for \(Q\) under \(B\). Below we first inductively construct a query \(Q_{\xi}\) from \(\xi\) such that
(a) \(\xi \equiv Q_{\xi} \equiv Q\), where \(Q \equiv_{B} \xi\) means that \(Q(D) = \xi(D)\) for any database \(D \models B\), and
(b) \(Q_{\xi}\) satisfies the condition on \(Q\) in Lemma 3. We then show that \(Q\) also satisfies the condition when \(Q_{\xi}\) satisfies it, and thus Lemma 3 holds.

Construction of \(Q_{\xi}\). We construct query \(Q_{\xi}\) by induction on the structure of \(\xi\) as follows:

o If \(\xi = \{c\}\), then \(Q_{\xi} = \{c\}\).

o If \(\xi = \sigma_{c}(\xi')\) (resp. \(\pi_{Y}(\xi')\)), then \(Q_{\xi} = \sigma_{c}(Q_{\xi'})\) (resp. \(\pi_{Y}(Q_{\xi'})\)), where \(Q_{\xi}\) is the query constructed for \(\xi\).

o If \(\xi = \sigma_{\xi'}(\xi')\), then \(Q_{\xi} = \sigma_{\xi}(Q_{\xi'} \land_{X} R(X,Y,Z))\).

By induction on the structure of \(\xi\), one can readily verify that for each relation \(R\) in \(Q_{\xi}\), there exists a bag constraint \(\varphi = R(X \to Y,N)\in B\) such that \(X^{Q_{\xi}}_{R} \subseteq XY \subseteq \text{cov}(Q_{\xi},B)\), i.e., \(Q_{\xi}\) satisfies the condition of Lemma 3.

Query \(Q\) satisfies the condition. We next show that \(Q\) satisfies the condition in Lemma 3 when \(Q_{\xi}\) does. Since for each bag constraint \(\varphi = R(X \to Y,N)\in B\), \(N \geq |Q|\), from \(Q_{\xi} \equiv_{B} Q\), one can verify that \(Q_{\xi} \equiv Q\). Thus there exists a homomorphism \(\rho\) from \(Q_{\xi}\) to \(Q^{B}\). Moreover, since \(Q\) is self-join free, each relation schema \(R\) (i.e., relation atom) has at most one occurrence in \(Q\). Then no relation atom in \(Q\) can be removed without changing \(Q\). Thus, \(Q\) is minimal (an SPC query is minimal if it has no redundant relation atoms[29]). Hence for each relation \(R\) in \(Q\), there must exist a relation \(R_{\xi}\) in \(Q_{\xi}\) such that \(\rho(R) = R_{\xi}\), and moreover, \(X^{Q_{\xi}_{\xi}}_{R_{\xi}} \subseteq \rho(X^{Q_{\xi}}_{R})\). Hence \(X^{Q_{\xi}_{\xi}}_{R} \subseteq \rho(\text{cov}(Q_{\xi},B)) \subseteq \text{cov}(Q,B)\). That is, \(Q\) also satisfies the condition of Lemma 3.

(\(\Leftarrow\)) Assume that \(Q\) satisfies the condition of Lemma 3. We construct a 3-step bounded query plan \(\xi\) under \(B\) for \(Q\):
(a) it has a bounded sub-plan \(\xi_{R}\) for each relation \(R\) in \(Q\) that fetches all attribute values needed for answering \(Q\);
(b) it combines the attribute values for each relation \(R\) in \(Q\) via a bounded sub-plan \(\xi_{R}\) such that each (partial) tuple fetched and kept for \(R\) is guaranteed to draw values from the same tuple in \(D\); and
(c) it finally carries out operations in \(Q\) over \(\xi_{R}\) for each relation \(R\) in \(Q\).

To show such a plan \(\xi\) exists for \(Q\) under the condition of Lemma 3, we only need to prove the following two properties:

(1) all necessary attribute values for answering \(Q\) from each relation \(R\) in \(Q\) can be retrieved by \(\xi_{R}\) in step (a), and
(2) their combinations can be restored by \(\xi_{R}\) in step (b).

Proof of (1). We prove (1) by constructing such a bounded plan \(\xi_{R[A]}\) for each attribute \(A \in X^{Q}_{R}\). Note that only attributes in \(X^{Q}_{R}\) are needed for answering \(Q\). The plan \(\xi_{R[A]}\) is constructed by translating the proof that witnesses \(A \in \text{cov}(Q,B)\). More specifically, since the condition of Lemma 3 holds for \(Q\) and \(B\), for any attribute \(A\) of \(X^{Q}_{R}\) such that \(A \in \text{cov}(Q,B)\), there must exist a se-
sequence of applications of rules (a)–(c) that defines \( \text{cov}(Q, B) \) such that
\[
l: \text{cov}_0 \xrightarrow{r_1} \text{cov}_1 \xrightarrow{r_2} \cdots \xrightarrow{r_n} \text{cov}_n
\]
where \( \text{cov}_0 = \emptyset \), \( A \in \text{cov}_m \), step \( i \) expands \( \text{cov}_{i-1} \) by applying rule \( r_i \) from one of the rules (a)–(c) for defining \( \text{cov}(Q, B) \) given earlier. We translate \( l \) into a bounded plan:
\[
\xi : \xi_0, \cdots, \xi_n
\]
where \( \xi_0 \) is empty; \( \xi_i \) is derived from \( \xi_0, \cdots, \xi_{i-1} \) based on step \( \text{cov}_{i-1} \xrightarrow{r_i} \text{cov}_i \), as follows:

(i) if \( r_i \) is rule (a) for a constant \( c \) in \( X^Q_0 \), then \( \xi_i = \{c\} \);

(ii) if \( r_i \) is rule (b) with \( A = B \) such that \( A \in \text{cov}_{i-1} \), \( B \notin \text{cov}_{i-1} \) and \( B \in \text{cov}_i \), then \( \xi_i = \xi_{i-1} \);

(iii) if \( r_i \) is rule (c), then \( \xi_i = \text{fetch}(\xi_{j_1} \cdots \xi_{j_k} \cup r, q) \), where \( \xi_{j_1}, \cdots, \xi_{j_k} \) are the bounded plans that fetch attributes in \( X(j_1, \cdots, j_k) < i \).

By the construction, \( \xi \) is a bounded plan that fetches all \( A \) attribute values for \( Q \). Note that each sub-plan \( \xi_i \) is bounded since it does not involve relation scans.

Proof of (2). Plan \( \xi_R \) is constructed with \( \xi_A \) for each \( A \in X^Q_B \) by (i) \( T_1 = \xi_A \cup \cdots \cup \xi_{A_i} \cup r \), where \( A_i \)'s \( (i \in [1, |X|]) \) range over all attributes in \( X \), and \( A_i \) is the plan generated above for fetching \( A_i \) values; and (ii) if \( X^Q_B \subseteq XY \) for \( \varphi = R(X \to Y, N) \in B \), then \( \xi_R = \text{fetch}(T_1, \varphi) \). Since \( \xi_A \) is a bounded plan for each \( A \in X^Q_B \), \( \xi_R \) is bounded under \( B \).

Hence, when the condition of Lemma 3 holds for \( Q \) and \( B \), \( \xi \) constructed above is a bounded plan for \( Q \) under \( B \).

(II) NP cases. One might want to lift the restriction of condition (b) on the queries in \( C^P \). This covers all SPC queries, including those with self-joins. However, the bounded evaluability analysis becomes harder unless \( P = NP \).

Class \( C^{NP} \). Denote by \( C^{NP} \) the set of pairs of bag access schemas \( B \) and SPC queries \( Q \) such that for each bag constraint \( \varphi = R(X \to Y, N) \), \( N > |Q| \). We have the following.

Theorem 4. For any bag access schema \( B \) and SPC \( Q \),

(a) it is in PTIME to decide whether \( (B, Q) \) is in \( C^{NP} \); and

(b) it is NP-complete to decide whether \( Q \) is boundedly evaluable under \( B \) if \( (B, Q) \) is in \( C^{NP} \).

Proof. Statement (a) is immediate. To prove statement (b), we first give a sufficient and necessary condition for query \( Q \) to be boundedly evaluable under \( B \) for any \( (B, Q) \in C^{NP} \). Based on the characterization, we then show that checking bounded evalubility for \( C^{NP} \) is NP-complete.

Let \( Q_m \) be the minimal equivalent query of \( Q \), i.e., the minimized version of \( Q \), which can be obtained by removing all redundant relations (see [2] for details). For an SPC query \( Q \), there exists a unique minimal equivalent query up to isomorphism[2]. Along the same lines as the proof of Lemma 3, one can verify the following for cases in \( C^{NP} \).

Lemma 5. For any \((B, Q) \in C^{NP}, \) \( Q \) is boundedly evaluable under \( B \) if and only if for each relation atom \( R \) in \( Q_m \), there exists \( \varphi = R(X \to Y, N) \in B \) such that \( X^Q_B \subseteq XY \subseteq \text{cov}(Q_m, B) \).

Based on this, we prove that deciding whether \( Q \) is boundedly evaluable under \( B \) is NP-complete for \((B, Q) \in C^{NP} \).

Upper bound. We give an NP algorithm as follows:

(a) convert \( Q \) into its tableau representation \((T_Q, u)[2]\);

(b) guess a sub-query \( Q' = (T, u) \) of \( Q \) such that \( T' \subseteq T_Q \) and a mapping \( \rho \) from \( T_Q \) to \( T' \);

(c) check (i) whether \( \rho \) is a homomorphism from \((T_Q, u)\) to \((T, u)\) and (ii) whether for each relation atom \( R \) in \( Q' \), there exists \( \varphi = R(X \to Y, N) \in B \) such that \( X^Q_B \subseteq XY \subseteq \text{cov}(Q; B) \); return “Yes” if so.

The algorithm is correct since if conditions (i) and (ii) of step (c) hold on \( Q' \), they must also hold on the minimal equivalent query \( Q_m \) of \( Q \). Indeed, \( \rho \) also determines a homomorphism from \((T_Q, u)\) to \((Q_m, u)\) since \( Q_m \) is a minimal equivalent query of \( Q \), i.e., \( T_{Q_m} \subseteq T_Q \subseteq T_Q \); therefore, if condition (ii) holds on \( Q' \), for homomorphism \( \rho \) it must also hold on \( Q_m \), i.e., the condition of Lemma 5 applies to \( Q \) and \( B \). Thus, by Lemma 5, \( Q \) is boundedly evaluable. The algorithm is in NP since step (a) is in PTIME, and step (c) is in PTIME in \(|Q|, |Q|, |\rho|, \) and \(|B| \) while \(|Q'| \leq |Q| \) and \( \rho = O(|Q|) \). Here \(|Q| \) is the size of \( Q \), i.e., the number of attributes and aggregate fields in \( Q \); \(|B| \) is the total length of bag constraints in \( B \). Lower bound. To show that the problem is NP-hard, we consider the following problem, denoted by MINQC.

○ Input: A relation schema \( R \) and an SPC query \( Q \) over \( R \).

○ Question: Is \( Q \) a minimal equivalent query of \( R \)? It is easy to verify that MINQC is \( coNP \)-complete by reduction from 3-COLORABILITY, which is \( NP \)-complete[20].

Lemma 6. Problem MINQC is \( NP \)-complete.

We show that the bounded evaluability problem for \( C^{NP} \) is NP-hard by reduction from the complement of MINQC.

Given an instance of MINQC, i.e., a relation schema \( R(A_1, \cdots, A_m) \) and an SPC query \( Q \) over \( R \), we construct a database schema \( R' \), an SPC query \( Q' \) over \( R' \) and a bag access schema \( B \). We show that \( Q \) is not minimal if and only if \( Q' \) is boundedly evaluable under \( B \).

(1) Database schema \( R' \) consists of a single relation schema \( R'(A_1, \cdots, A_n, B_1, \cdots, B_{n-1}) \), where \( n \) is the number of relation atoms that appear in query \( Q \).

Intuitively, \( R \) extends \( R \) with additional attributes
\[
B_i, \cdots, B_{n(n-1)}. \text{ As will be shown later, together with } Q', \text{ such new attributes will be used as join attributes to pairwise connect the } n \text{ relation atoms of } Q \text{ in } Q'.
\]

(2) Query \( Q' \) is derived from \( Q \) as follows:

- \( \phi_1 = B_i \) retains the same number of joins and relation atoms as \( Q \), such that each relation atom \( R_i (i.e., renaming of relation schema } R \in \mathcal{R}) \) is replaced with \( R' \) (i.e., renaming of relation schema \( R' \in \mathcal{R}' \)); and
- \( \phi_2 = R'_{i\cdots i} \), add equality \( R'[B_i] = R'_i[B_i] \to C \), where

\[
p = \frac{n(i-1)}{2} + (j-i).
\]

Intuitively, \( C \) preserves all selection conditions of \( Q \) and additionally joins each pair of the \( n \) relation atoms on a dedicated attribute \( B_i(i \in [1, \frac{n(n-1)}{n}]) \): (a) for each \( B_i \), there exist exactly two relation atoms \( R_i' \) and \( R_j' \) such that \( R_i'[B_i] = R'_j[B_i] \); and (b) for each \( R_i' \) and \( R_j' \), there exists exactly one attribute \( B_k \) such that \( R'_i[B_i] = R'_j[B_k] \).

(3) The bag access scheme \( B \) consists of \( \frac{n(n-1)}{2} - 1 \) constraints. Let \( W \) be the set of all attributes of \( A_1, \cdots, A_m \) such that they appear in the selection/join conditions or the top-level projection attributes in \( Q \). Then \( B \) consists of

\[
\begin{align*}
\phi_1 &= R'_{i\cdots i} [W \to B_{B1} \cdots B_{B\frac{n(n-1)}{n} 1}, N], \\
\phi_2 &= R'_i [W \to B_{B1} \cdots B_{B\frac{n(n-1)}{n} 1}, N], \\
&\vdots \\
\phi_{\frac{n(n-1)}{2}} &= R'_{i\cdots i} [W \to B_{B1} \cdots B_{B\frac{n(n-1)}{n} 1}, N].
\end{align*}
\]

We next show that query \( Q \) is not minimal if and only if \( Q \) is boundedly evaluable under \( B \).

(⇒) Assume that \( Q \) is not minimal. Then none of the \( n \) relation atoms in \( Q \) can be removed by minimizing \( Q' \). Hence all \( \frac{n(n-1)}{2} \) attributes \( B_1, \cdots, B_{\frac{n(n-1)}{n}} \), together with \( W \), have to be contained in \( XY \) for some \( \varphi = R'(X \to Y, N, m) \) in \( B \) by Lemma 5. This yields \( |W| + \frac{n(n-1)}{2} \) attributes. This is impossible since for any \( \varphi_i \in B, \varphi_i \) contains \( |W| + \frac{n(n-1)}{2} - 1 \) attributes only by its definition above.

(⇐) Assume that \( Q' \) is boundedly evaluable under \( B \).

Since each \( \varphi_i \in B \) contains \( |W| + \frac{n(n-1)}{2} - 1 \) attributes, by Lemma 5, \( X_{R''}^{Q''} \) must contain at most \( |W| + \frac{n(n-1)}{2} - 1 \) attributes for each relation atom \( R'' \) in \( Q' \), where \( Q'' \) is the minimal equivalent query of \( Q' \).

Since \( X_{R''}^{Q''} \) contains \( |W| + \frac{n(n-1)}{2} \) attributes, query \( Q' \) is not minimal.

Remark 2. Despite its intractability, checking the bounded evaluable for CNF is feasible in practice by Lemma 5. Indeed, there have been effective algorithms for minimizing SPC queries, i.e., computing \( Q_m \) for \( Q' \), and the size of \( Q \) is typically small. Taking one of these algorithms as an oracle for computing \( Q_m \), one can still efficiently check the bounded evaluability of generic SPC queries: first minimize \( Q \), yielding \( Q_m \), and then check whether \( Q_m \) and \( B \) satisfy the condition of Lemma 5 in PTIME in \( |Q_m| \) and \( |B| \).

4 Effective syntax

In this section, we propose an effective method to check the bounded evaluability of generic RA\(_{aggr} \) queries. We show that while the problem is undecidable (Theorem 1), there exists an effective syntax for boundedly evaluable RA\(_{aggr} \) queries, which reduces the problem to syntactic checking (Section 4.1). In addition, we identify two practical subclasses of RA\(_{aggr} \) queries and provide their effective syntax. In particular, we give one for RA and show that it covers more bounded queries than the one given in [6] (Section 4.2).

4.1 An effective syntax for RA\(_{aggr} \)

Under an access schema \( S \), an effective syntax for boundedly evaluable queries of \( L \) (\( L \) refers to, e.g., RA or RA\(_{aggr} \)) is a subclass \( L_{B}^{\mathcal{L}} \) of \( L \) such that for any \( Q \in L \),

- (a) if \( Q \) is boundedly evaluable, then there exists a query \( Q' \in L_{B}^{\mathcal{L}} \) such that \( Q \equiv B Q' \);
- (b) every query \( Q \in L_{B}^{\mathcal{L}} \) is boundedly evaluable; and
- (c) it is in PTIME in the size \( |Q| \) of query \( Q \) and the length \( |B| \) of constraints in \( B \) to check whether \( Q \in L_{B}^{\mathcal{L}} \).

Here \( Q \equiv B Q' \) if \( Q(D) = Q'(D) \) for all databases \( D \models B \).

Intuitively, the effective syntax reduces the problem of deciding the bounded evaluability of \( L \) queries to syntactic checking of \( L_{B}^{\mathcal{L}} \). Indeed, every boundedly evaluable \( L \) query can find an equivalent query in \( L_{B}^{\mathcal{L}} \) under \( B \). Hence, we can safely settle with queries in \( L_{B}^{\mathcal{L}} \), since \( L_{B}^{\mathcal{L}} \) can express, up to equivalence, all boundedly evaluable \( L \) queries.

Remark 3. To some extent, the development of effective syntax \( L_{B}^{\mathcal{L}} \) is analogous to the study of range-safe queries for relational calculus. Indeed, the problem for checking the “safety” of relational calculus queries is also undecidable[3]. Despite this, range-safe queries are supported by commercial DBMS, by making use of an effective syntax of range-safe relational calculus queries. We follow the same approach to dealing with the bounded evaluable of RA\(_{aggr} \) queries.

Below we develop such an effective syntax, denoted by \( L_{B} \) for RA\(_{aggr} \) queries that are boundedly evaluable under \( B \).

The class \( L_{B} \). In a nutshell, we characterize \( L_{B} \) with three sets: \( BA(Q, B), BR(Q, B) \) and \( BQ(Q, B) \). Informally, under a bag access schema \( B \), for an RA\(_{aggr} \) query \( Q \),

- \( BA(Q, B) \) contains attributes (e.g., \( A \) ) and aggregates (e.g., \( \sum(A) \) ) of \( Q \) whose values can be fetched via...
B;

\( \circ \text{BR}(Q, B) \) consists of relations in Q whose partial tuples that are needed to answer Q can be reconstructed from the fetched values for attributes in BA(Q, B); and

\( \circ \text{BQ}(Q, B) \) contains boundedly evaluable sub-queries of Q.

An RA\textsubscript{aggr} query Q is included in \( L_B \) if Q \( \in \text{BQ}(Q, B) \).

Intuitively, these sets characterize RA\textsubscript{aggr} queries Q for which the values of all attributes necessary for answering Q can be "deduced" from constants in Q, via joins and fetch under access schema B. Such attributes participate in RA\textsubscript{aggr} operations of Q, and are referred as the nontrivial attributes of query Q. The class of such queries makes an effective syntax for boundedly evaluable RA\textsubscript{aggr} queries.

More specifically, sets BA, BR and BQ are defined in a mutual recursive way using rules given in Fig.1, with notations explained in Table 1. Intuitively, (1) rule \( \gamma_1 \) of Fig.1 includes constant attributes \( X_0 \) (see Table 1) in BA(Q, B); (2) \( \gamma_2 \) propagates value from attributes and aggregate fields to join attributes; (3) \( \gamma_3 \) specifies value propagation via fetch; (4) \( \gamma_4 \) says that if a sub-query is boundedly evaluable, then its output attributes and aggregate fields can also be fetched; (5) \( \gamma_5 \) adds a relation atom R to BR(Q, B) only when the partial tuples of R can be reconstructed from combinations of the fetched values; and (6) \( \gamma_6 \) says that a sub-query is boundedly evaluable if all its relations can be correctly fetched. As shown in Table 1, \( \Sigma_Q \) denotes the set of equality predicates embedded in the selection or join conditions of Q, and \( \Sigma_Q \models A = B \) denotes that equality A = B can be deduced from \( \Sigma_Q \) by the transitivity of the equality relation.

**Example 7.** Recall query Q\textsubscript{2} from Example 2 and bag access schema B\textsubscript{1} from Example 4. We show that Q\textsubscript{2} \( \in L_B \).

(1) Initially, by rule \( \gamma_1 \), BA(Q\textsubscript{2}, B\textsubscript{1}) includes f.uid and c.pty, where f (resp. c) are shorthands for friend (resp. checkin).

(2) By rule \( \gamma_2 \) and \( \varphi_1 \), BA(Q\textsubscript{2}, B\textsubscript{1}) further includes f.fid.

(3) Since \( \Sigma_Q \models f.\text{fid} = c.\text{uid} \), by f.fid \( \in \text{BA}(Q_2, B_1) \), we have that BA(Q\textsubscript{2}, B\textsubscript{1}) includes c.uid by rule \( \gamma_2 \).

(4) By \( \gamma_6 \), f \( \in \text{BR}(Q_2, B_1) \) because both f.uid and f.fid are in BA(Q\textsubscript{2}, B\textsubscript{1}) and are attributes of \( \varphi_1 \). Similarly, c \( \in \text{BR}(Q_2, B_1) \) because of \( \varphi_2 \). Note that only c.uid and c.pty are nontrivial attributes of Q in relation c and they are both in BA(Q\textsubscript{2}, B\textsubscript{1}).

(5) By \( \gamma_6 \), sub-query Q\textsubscript{3} and query Q\textsubscript{2} itself are in BQ(Q, B).

As another example, recall query Q\textsubscript{6} from Example 5 and B\textsubscript{2} from Example 4. We next show that Q\textsubscript{6} \( \in L_B \).

(1) One can readily deduce that BA(Q\textsubscript{6}, B\textsubscript{2}) includes B and C by using \( \gamma_1 \) and \( \gamma_3 \), and that BR(Q\textsubscript{6}, B\textsubscript{2}) includes R with \( \gamma_7 \).

(2) By \( \gamma_6 \), BQ(Q\textsubscript{6}, B\textsubscript{2}) includes sub-query Q of Q\textsubscript{6}.

(3) Hence, further by \( \gamma_4 \) we have that Z\textsubscript{Q} \( \models \text{BA}(Q_6, B_2) \), where Z\textsubscript{Q} is the output of Q\textsubscript{7}, i.e., an aggregate field.

(4) By \( \gamma_2 \), we know that BA(Q\textsubscript{6}, B\textsubscript{2}) includes F since \( \sigma_{Q_6} \models Z_{Q_7} = F \) and F is not an aggregate field.

(5) Thus, by \( \gamma_6 \) and \( \gamma_6 \), S \( \in \text{BR}(Q_6, B_2) \) and Q\textsubscript{6} \( \in \text{BQ}(Q_6, B_2) \).

Note that Q\textsubscript{6} \( \notin L_B \). However, Q\textsubscript{4} \( \equiv_{B_2} Q_6 \), Q\textsubscript{6} \( \in L_B \) and Q\textsubscript{6} is boundedly evaluable under B\textsubscript{2}. Similarly, for Q\textsubscript{6}, Q\textsubscript{61} and B\textsubscript{1} of Example 6, one can verify that Q\textsubscript{6} \( \notin L_B \), but Q\textsubscript{63} \( \in L_{B_3} \) and Q\textsubscript{63} is boundedly evaluable under B\textsubscript{3}.

\[ \text{To reduce notations, we assume } w.l.o.g. \text{ that selection conditions are conjunctive, i.e., } \forall \text{ in selection predicates is reduced using } \cup. \]
We next show that $\mathcal{L}_B$ is indeed an effective syntax for boundedly evaluable RA$_{agg}$ queries under $B$.

**Theorem 7.** Under any bag access schema $B$, $\mathcal{L}_B$ is an effective syntax for boundedly evaluable RA$_{agg}$ queries.

**Proof.** We show below that $\mathcal{L}_B$ has properties (a) and (b) of an effective syntax, by proving the following lemmas. We will constructively prove property (c) in Section 5.2.

(i) For any bounded plan $\xi$ under $B$, there is $Q \equiv_B \xi$ in $\mathcal{L}_B$.

(ii) For any $Q \in \mathcal{L}_B$, there is plan $\xi \equiv_B Q$ bounded under $B$.

These suffice. Indeed, for any $Q$ that is bounded under $B$, by definition there must exist a plan $\xi Q$ bounded under $B$; hence by (i), there exists $Q' \equiv_B \xi Q$ in $\mathcal{L}_B$. On the other hand, if $Q \equiv_B Q' \in \mathcal{L}_B$, by (ii), $Q'$ has a bounded plan $\xi' \equiv_B Q' \equiv_B Q$, i.e., $Q$ is also boundedly evaluable under $B$. Hence, $\mathcal{L}_B$ has properties (a) and (b) of an effective syntax.

We next prove the two lemmas.

**Proof of (i).** We prove it by induction on the structure of $\xi$.

**Base case.** When $\xi = \{c\}$ or $\emptyset$, by definition $\xi$ itself is in $\mathcal{L}_B$.

**Induction.** We consider the structure of $\xi$.

(i) $\xi$ is $g$-by-$\ell(X, agg(V))$. By the induction hypothesis, there exists a query $Q' \equiv_B \ell \xi$ such that $Q' \in \mathcal{L}_B$. Consider $Q = g$-by-$\ell(Q', X, agg(V))$. By the definition of $\mathcal{L}_B$, all relations in $Q'$ are in $BR(Q', B) \subseteq BR(Q, B)$ (since $Q$ and $Q'$ share the same nontrivial attributes). Hence $Q \in BQ(Q, B)$ by rule $\gamma_6$. That is, $Q \in \mathcal{L}_B$. Obviously, $Q \equiv_B \xi$.

The cases for $\xi = \pi_y(\xi')$, $\sigma_C(\xi')$, $\xi_1 \times \xi_2$, $\xi_1 \cup \xi_2$, $\xi_1 - \xi_2$ are similar and can be verified along the same lines.

(ii) $\xi$ is fetch($\ell', \varphi$) with $\varphi = R[X \rightarrow Y, N]$. By the induction hypothesis, there exists query $Q' \equiv_B \ell' \xi$ such that $Q' \in \mathcal{L}_B$. Consider $Q = \tau_{R[X,Y]}(Q' \overset{\Lambda}{\rightarrow}_{Z_{20} \ldots Z_{2k}} R)$. By the semantics of fetch, $Q \equiv_B \xi$. We next show that $Q \in \mathcal{L}_B$. Since $Q' \in \mathcal{L}_B$, $Q' \in BQ(Q', B)$. Hence by rule $\gamma_4$, $Q'[X] \subseteq BA(Q', B) \subseteq BA(Q, B)$. Further by $\gamma_2$, $R[X] \subseteq BA(Q, B)$. Hence by $\gamma_6$, $Q \equiv_B BQ(Q, B)$. That is, $\xi \in \mathcal{L}_B$.

**Proof of (ii).** Since $Q \in \mathcal{L}_B$, by the definition of $\mathcal{L}_B$ there must exist a proof $\ell Q$ consisting of applications of rules in Fig.1 that deduces $Q \in BQ(Q, B)$, i.e., a sequence of the form

$$(BA_0, BR_0, BQ_0) \overset{r_1}{\rightarrow} \cdots \overset{r_n}{\rightarrow} (BA_n, BR_n, BQ_n)$$

where (a) $r_i (i \in [1, n])$ is one of the rules in Fig.1; (b) $BA_0 = BR_0 = BQ_0 = \emptyset$; (c) for each step $i$, only one of $BA_i$, $BR_i$, and $BQ_i$ is changed in $BA_{i+1}$, $BR_{i+1}$, and $BQ_{i+1}$, respectively; and (d) $r_n$ is rule $\gamma_6$ that deduces $Q \in BQ(Q, B)$. We define the length of $\ell Q$ as the number $n$ of rules applied in $\ell Q$.

**Induction hypothesis.** We show that for a proof $\ell$ of length $i$,

- if $A \in BA_{i+1}$ but $A \notin BA_i$, values for $A$ that are necessary for answering $Q$ can be fetched via bounded plan under $B$;
- if $R \in BR_{i+1}$ but $R \notin BR_i$, then values from $R$ necessary for $Q$ can be fetched via bounded plans under $B$; and
- if $q \in BR_{i+1}$ but $q \notin BQ_i$, then the exact answers to sub-query $Q$ of $Q$ can be answered via bounded plans under $B$.

Note that for any $Q \in \mathcal{L}_B$, $Q$ must have a proof $\ell Q$ ending by including $Q \in BQ(Q, B)$. If the induction hypothesis holds, $Q$ must have a bounded plan under $B$, which proves lemma (ii).

We next prove it by induction on the length $l(\ell)$ of $\ell$.

**Base case.** When $l(\ell) = 1$, then rule $r_1$ can only be either (i) $\gamma_1$ of Fig.1, i.e., to include $A \in BA_1$ from selection $A = c$ of $Q$; or (ii) $\gamma_6$ of Fig.1, i.e., to include $R[Y]$ in $BA_1$ with $\varphi = R[\emptyset \rightarrow Y, N] \in B$. For (i), simply let $\xi_1 = \{c\}$. Then $\xi_1$ is a bounded plan that fetches all necessary values for $A$. For (ii), let $\xi_2[\varphi] = \{\emptyset, \varphi\}$. Then by the semantics of fetch, all values for $R[Y]$ that are necessary for answering $Q$ are fetched by $\xi_1[\varphi]$ (here we rename $Q$ beforehand such that there exist no duplicated attribute names).

**Induction.** Assume that the hypothesis holds for proofs of length at most $k$. Consider proof $\ell$ of length $k+1$. We discuss the last step $(BA_i, BR_i, BQ_i) \overset{r_{i+1}}{\rightarrow} (BA_{i+1}, BR_{i+1}, BQ_{i+1})$ of $\ell$.

(i) If $r_{i+1}$ is rule $\gamma_1$ with attribute $A = c$, then $A$ can be fetched in exactly the same way as the base case.

(ii) If $r_{i+1}$ is rule $\gamma_6$ that includes attribute $B$ in $BA_{i+1}$ with $A = B$, then attribute $A$ must be included in $BA_{i+1}$ at some steps prior to $k+1$. By the induction hypothesis, there must exist a bounded plan $\xi_A$ that fetches all necessary values for answering $Q$ except $B$. Hence $\xi_2 = \xi_A$ is also a bounded plan that fetches $B$ for $Q$ by the condition $A = B$.

(iii) If $r_{i+1}$ is rule $\gamma_3$ that includes $R[Y]$ in $BA_{i+1}$ with $R[X] \subseteq BA(Q, B)$ and $\varphi = R[X \rightarrow Y, N]$, then there exist steps $1, \ldots, i$, prior to $k+1$ that include $R[X_1], \ldots, R[X_n]$ in $BA$ such that $R[X_1] \cup \cdots \cup R[X_n] = R[X]$. Hence by the induction hypothesis, $R[X_i][j \in [1, p]]$ has bounded plan $\xi_{R[X_i]}$. Let $\xi_{R[X_i]} \be \Lambda^{\gamma_{j=1}} \xi_{R[X_i]}$, then $\xi_{R[X_i]}$ fetches all values for $R[X]$ that are necessary for answering $Q$. Hence, further by the semantics of fetch, $R[Y]$ has a plan fetch($\xi_{R[X]}, \varphi$) that retrieves all $R[Y]$-values needed for answering $Q$.

(iv) If $r_{i+1}$ is rule $\gamma_3$ that includes $Z_{20}$ in $BA_{i+1}$ from a subquery $Q_e$ of $Q$ that is included in $BQ_j$ at step $j < i + 1$, then by the induction hypothesis, there exists a plan $\xi_{Q_e}$ for $Q_e$ under $B$ that exactly answers $Q_e$. Hence
we can get values for its output attributes $Z_Q$, simply by $\xi_Q$. Note that since $\xi_Q$ is an exact plan for $Q$, the values for $Z_Q$ are guaranteed correct even when $Q$ is an aggregate subquery.

(v) If $r_{k+1}$ is $\gamma_0$ that includes $R$ in $BQ_{k+1}$ with $R[X]$ and $\varphi = R(X \rightarrow Y, N) \in B$, then by the induction hypothesis, there exist plans $\xi_1, \ldots, \xi_{r_{k+1}}$ to fetch all necessary values for $\xi_1, \ldots, \xi_{r_{k+1}}$ for $Q$, respectively, such that $\xi_1 \cup \cdots \cup \xi_{r_{k+1}} = \xi$, hence $R$ has a bounded $R[X,Y,\varphi]$ that fetches all necessary $R[X]$-values for $Q$. Since $R[X,Y]$ covers all nontrivial attributes of $R$ for $Q$, by $\xi R[X,Y] = fetch(\xi R[X], \varphi)$, we can fetch all combinations of $R[X]$-values that are needed for answering $Q$.

(vi) If $r_{k+1}$ is $\gamma_0$ that includes $Q_s$ in $BQ_{k+1}$, then all relations $R_1, \ldots, R_p$ of $Q_s$ have been included in $B$ in prior steps. Hence by the induction hypothesis, there exist $\xi_1, \ldots, \xi_p$ that fetch all values from $R_1, \ldots, R_p$, respectively, which are necessary for $Q$. Now construct plan $\xi_{r_{k+1}}$ by replacing each relation $R_i(i \in [1,p])$ in $Q_s$ with $\xi_i$. Then $\xi_{r_{k+1}}$ must be a query plan for $Q_s$ of $Q$ since all necessary value combinations can be retrieved from $D$ via $\xi_{r_{k+1}}, i \in [1,p]$, and $Q_s$ then checks and combines values exactly the same as on $D$.

Hence the hypothesis holds for proofs of length $k+1$. □

4.2 Special cases

These are two important sub-classes of $RA_{agg}$: (1) RA consists of $RA_{agg}$ queries without aggregation; and (2) $RA_{agg}^0$ is the class of $RA_{agg}$ queries in which group-by aggregation, if it exists, only appears at the top-level (final operation). It is common to find RA and $RA_{agg}^0$ in practice.

We provide effective syntax for boundedly evaluable RA and $RA_{agg}^0$. Queries. Denote by $\mathcal{L}_B[RA]$ and $\mathcal{L}_B[RA_{agg}^0]$ the class of RA and the class of $RA_{agg}^0$ queries that are in $\mathcal{L}_B$, respectively. They yield effective syntax for RA and $RA_{agg}^0$.

Corollary 8. For any access schema $B$, (1) $\mathcal{L}_B[RA]$ is an effective syntax for RA queries bound by $B$; (2) $\mathcal{L}_B[RA_{agg}^0]$ is an effective syntax for $RA_{agg}^0$ queries bound by $B$.

Proof. Since it is in PTIME to check whether a query is in $\mathcal{L}_B$ and every query in $\mathcal{L}_B$ has a bounded plan under $B$, to show that $\mathcal{L}_B[RA]$ and $\mathcal{L}_B[RA_{agg}^0]$ are effective syntax for boundedly evaluable RA and $RA_{agg}^0$ queries, respectively, it suffices to show that for any boundedly evaluable RA $Q_1$ and $RA_{agg}^0$ $Q_2$, there exist $Q_1 \in \mathcal{L}_B[RA]$ and $Q_2 \in \mathcal{L}_B[RA_{agg}^0]$ such that $Q_1 \equiv_B Q_1$ and $Q_2 \equiv_B Q_2$. This is verified along the same lines as the proof of Lemma (1) for Theorem 7 above, by showing that every boundedly evaluable (resp. $RA_{agg}^0$) plan has an equivalent query in $\mathcal{L}_B[RA]$ (resp. $\mathcal{L}_B[RA_{agg}^0]$).

There are close connections between $RA_{agg}^0$ and RA regarding their effective syntax: any effective syntax for boundedly evaluable RA queries also gives us an effective syntax for boundedly evaluable RA queries, and vice versa. For any class $\mathcal{L}$ of $RA_{agg}$ queries $Q = gpBy(Q’, X, agg(V))$, denote by (a) $\mathcal{L}_B$ the class of RA queries $Q$ that are sub-queries embedded in $RA_{agg}$ queries $Q$ in $\mathcal{L}$; and (b) $\mathcal{L}[RA]$ the class of RA queries in $\mathcal{L}$. Then we have the following.

Lemma 9. Under any bag access schema $B$, (1) RA is an effective syntax for boundedly evaluable RA if $\mathcal{L}$ is an effective syntax for boundedly evaluable RA;

(2) $\mathcal{L}$ is an effective syntax for boundedly evaluable RA,$RA_{agg}$ if $\mathcal{L}_B$ is an effective syntax for boundedly evaluable RA.

Proof. Lemma 9(1) can be verified by the definition of effective syntax. We focus on Lemma 9(2) here (the proof for Lemma 9(1) is simpler). By the definition of boundedly evaluable queries, it is easy to show the following lemma: for any RA,$RA_{agg}$ query $Q = gpBy(Q’, X, agg(V))$, under $B$, $Q$ is boundedly evaluable if $Q’$ is boundedly evaluable.

We next use the lemma to prove Lemma 9(2). When $\mathcal{L}_B$ is an effective syntax for boundedly evaluable RA queries under $B$, consider the associated class $\mathcal{L}$ of $\mathcal{L}_B$. (1) First observe that all queries in $\mathcal{L}$ are also boundedly evaluable since RA queries in $\mathcal{L}_B$ are. (2) For any RA,$RA_{agg}$ query $Q = gpBy(Q’, X, agg(V))$ that is boundedly evaluable, by the lemma above, $Q_1$ is also boundedly evaluable under $B$. Hence, there exists $Q_1 \equiv_B Q’$. Hence $Q’ = gpBy(Q’, X, agg(V)) \equiv_B Q$ and $Q’ \in \mathcal{L}$ (since $Q’ \in \mathcal{L}_B$). Moreover, it is in PTIME to check whether a query $Q$ is in $\mathcal{L}$ by checking whether its embedded RA query $Q’$ is in $\mathcal{L}_B$, in PTIME. From (1), (2) and (3) above, Lemma 9(2) follows. □

By Lemma 9, one can easily extend an effective syntax for boundedly evaluable RA queries, e.g., covered RA in[6], to an effective syntax for boundedly evaluable RA queries.

One might think that such an extension is also possible for RA,$RA_{agg}$. However, when group-by aggregation is nested with other RA,$RA_{agg}$ operators, a convenient extension is beyond reach. It is much harder for RA,$RA_{agg}$ to characterize propagation of values from aggregate sub-queries to other relations, or to cover all boundedly evaluable queries up to equivalence. Nonetheless, $\mathcal{L}_B[RA]$ is more expressive than the class of covered RA of [6], which is also an effective syntax for RA.

Proposition 10. For any bag access schema $B$, the set of RA queries covered by $B$ is properly contained in $\mathcal{L}_B[RA]$.

Proof. One can verify that covered RA queries[6] can be expressed in $\mathcal{L}_B[RA]$ without rule $\gamma_0$. Hence it is a subclass of $\mathcal{L}_B$. To see it is a proper subclass, consider an
In this section, we show how to extend DBMS with the functionality of bounded evaluation. We first present such a framework (Section 5.1). We then provide algorithms underlying the framework, for checking the bounded evaluability (Section 5.2) and generating bounded plans (Section 5.3).

5.1 A Framework of bounded evaluation

The framework, referred to as BEAS, is shown in Fig. 2. Given an application that involves queries over instances of a database schema $\mathcal{R}$, BEAS works as follows.

- **Offline preprocessing.** As shown in C1 of Fig. 2, as offline preprocessing, BEAS discovers a bag access schema $\mathcal{B}$ from (sample) instances of $\mathcal{R}$, builds indices of $\mathcal{B}$ on the database $\mathcal{D}$ of $\mathcal{R}$ in use, and maintains $\mathcal{B}$ in response to updates to $\mathcal{D}$.

- **Online processing.** When a user poses an RA$_{agg}$ query $Q$ on $\mathcal{D}$, BEAS first checks whether $Q$ is boundedly evaluable under $\mathcal{B}$ (C2). If so, it generates a bounded query plan $\xi$ for $Q$ under $\mathcal{B}$ (C3), which is interpreted as an SQL query $Q_\xi$ and hence can be directly executed by the underlying DBMS on a bounded dataset $\mathcal{D}_Q$ identified by plan $\xi$ (C4). If $Q$ is not boundedly evaluable, it generates a query plan $\xi$ for $Q$ that is partially bounded, to make maximal use of access constraints in $\mathcal{B}$ (C5). The (partially) bounded plans are optimized and executed by DBMS (C4).

Note that the BEAS framework does not need to change the underlying DBMS. Indeed, it interacts with the DBMS via SQL only. Hence, BEAS can be built on top of any existing DBMS, providing a bounded evaluation capacity.

BEAS can also compute approximate answers to unbounded queries under constrained resources, and offers deterministic accuracy guarantees under access schema [17]. We focus on computing exact answers in this paper.

Below we develop algorithms for components C2 and C3 of BEAS in Section 5.2 and Section 5.3, respectively.

5.2 Checking bounded evaluability

We next develop a practical algorithm for component C2 of BEAS. Under a bag access schema $\mathcal{B}$, given an RA$_{agg}$ query $Q$, it decides whether $Q$ is boundedly evaluable.

To do this, we first check whether $Q$ and $\mathcal{B}$ fall into the two classes of special cases, i.e., $\mathcal{O}^P$ or $\mathcal{O}^{NP}$, in PTIME. If so, their bounded evaluability can be decided efficiently as shown in the proofs of Theorems 2 and 4. Otherwise, we check whether $Q$ is in the effective syntax $\mathcal{L}_B$ for RA$_{agg}$ (Section 4). Below we give a PTIME algorithm for this.

**Algorithm BEChk.** The algorithm, denoted by BEChk, is shown in Algorithm 1. It first sets $BA(Q, \mathcal{B})$, $BR(Q, \mathcal{B})$ and $BQ(Q, \mathcal{B})$ to $\emptyset$. It then iteratively updates $BA(Q, \mathcal{B})$, $BR(Q, \mathcal{B})$ and $BQ(Q, \mathcal{B})$ using the rules in Fig. 1. In each iteration, it

(a) first computes $BA(Q, \mathcal{B})$ using $\gamma_1$, $\gamma_2$ and $\gamma_3$ (line 3);

(b) then updates $BR(Q, \mathcal{B})$ using $\gamma_4$ (line 4); and

(c) it finally updates $BQ(Q, \mathcal{B})$ using $\gamma_5$ (line 5).

The iteration continues until $BQ(Q, \mathcal{B})$ can no longer be updated (line 6). It returns “Yes” if $Q \in BQ(Q, \mathcal{B})$, and “No” otherwise (lines 7–8). In each iteration, steps (b) and (c) are straightforward. Below we discuss step (a) in more details.

**Algorithm 1.** BEChk

**Input:** Relational schema $\mathcal{R}$, RA$_{agg}$ $Q$ and bag access schema $\mathcal{B}$ over $\mathcal{R}$.

**Output:** “Yes” (“No”) if $Q$ is (is not) boundedly evaluable under $\mathcal{B}$.

1. $BA(Q, \mathcal{B}) \leftarrow \emptyset$; $BR(Q, \mathcal{B}) \leftarrow \emptyset$; $BQ(Q, \mathcal{B}) \leftarrow \emptyset$; flag $\leftarrow$ true;

2. while flag = true do

3. compute $BA(Q, \mathcal{B})$ using $\gamma_1$, $\gamma_2$, $\gamma_3$ (recall $\gamma_i$ in Table 1)

4. compute $BR(Q, \mathcal{B})$ using $\gamma_5$;

5. compute $BQ(Q, \mathcal{B})$ using $\gamma_6$;

6. if $BQ(Q, \mathcal{B})$ is not changed then flag $\leftarrow$ false;

7. if $Q \in BQ(Q, \mathcal{B})$ then return “Yes”;

8. else return “No”;

Computing $BA(Q, \mathcal{B})$ (line 3 of Algorithm 1). In each
iteration, \( \text{BA}(Q, B) \) is updated in two steps, as follows.

(1) Building universal relation. We first build a “universal schema” \( U_Q \) of \( Q \) w.r.t. \( BQ(Q, B) \) by mapping attributes and aggregate fields of \( Q \) to attributes of \( U_Q \) via a mapping function \( \rho \). For any two attributes \( R[A] \) and \( S[B] \) of \( Q \), \( \rho(R[A]) = \rho(S[B]) \) if and only if \( \Sigma_Q \vdash R[A] = S[B] \) in the selection condition of \( Q \). For aggregate field \( \text{agg}(A) \) and attribute \( R[B] \), \( \text{agg}(A) = \rho(R[B]) \) only when \( \text{agg}(A) \) is in \( Z_Q \) (recall Table 1) for some \( Q \in BQ(Q, B) \). Accordingly, bag constraints in \( B \) are also mapped onto to \( U_Q \) by \( \rho \).

(2) Computing fetch closure. We then reduce the computation of \( \text{BA}(Q, B) \) to the computation of fetch closures over \( U_Q \) with \( B \) w.r.t. \( \rho \). For a set \( W \) of attributes of \( U_Q \), its fetch closure, denoted by \( W^B \), is a set of attributes of \( U_Q \) such that

(i) \( W \subseteq W^B \);

(ii) if \( X' \subseteq W^B \) and \( \varphi = R(X \rightarrow Y, X') \in B \) such that \( \rho(R[X]) = X' \) and \( \rho(R[Y]) = Y' \), then \( Y' \subseteq W^B \); and

(iii) \( W^B \) contains nothing else.

Let \( W = \rho(X^B) \cup \rho(BA(Q, B)) \cup \bigcup_{Q \in BQ(B, B)} \rho(Z_Q) \). We set \( \text{BA}(Q, B) = \{ A \in X_Q \mid \rho(A) \in W^B \} \) (see \( Z_Q, X_Q \) in Table 1).

Example 8. Recall \( Q_2 \) from Example 2 and bag access schema \( B_1 \) from Example 4. Algorithm BEChk iteratively updates \( BA, BR \) and \( BQ \) for \( Q_2 \) and \( B_1 \), which are all \( \emptyset \) initially.

In the first iteration, BEChk starts by updating \( BA(Q_2, B_1) \) (line 3). To do this, it builds a universal relation \( U_{Q_2} = \{ f.uid, f.fid, c.cty, c.date \} \) via function \( \rho \) that maps \( c.uid \) to \( f.fid \) and keeps all other attributes intact (\( f \) and \( c \) stand for friend and checkin, respectively). Since \( \rho(X_{Q_2}) = \{ f.uid, c.cty \}, \rho(BA(Q_2, B_1)) = \emptyset \) and \( BQ(Q_2, B_1) = \emptyset \), BEChk sets \( W \) to \( \rho(X_{Q_2}) \) and computes the fetch closure \( W^B_{Q_2} \) of \( W \), yielding \( W^B_{Q_2} = \{ f.uid, f.fid, c.cty \} \). Hence it updates \( BA(Q_2, B_1) \) to \( \{ f.uid, f.fid, c.cty \} \) and \( BQ(Q_2, B_1) \) to \( \{ f.c, c \} \) since all nontrivial attributes of \( f \) and \( c \) are already in \( BA(Q_2, B_1) \) (line 4). BEChk finally updates \( BQ(Q_2, B_1) \) to \( \{ Q_3, Q_4 \} \) (line 5) and terminates in next iteration and returns “Yes”.

It gets more involved for \( Q_6 \) from Example 5 and \( B_2 \) from Example 4. In the first iteration, BEChk builds a universal schema \( U_{Q_6} = \{ A, B, C, E, F, W, Y' \} \) via a mapping function \( \rho \) that keeps attributes of \( R \) and \( S \) and maps aggregate field (i.e., the output) \( \text{sum}(y) \) of \( Q_7 \) to \( F \). Note that \( \rho \) does not map \( \text{sum}(y) \) to \( E \) although \( \Sigma_{Q_6} \vdash \text{sum}(y) = E \) since \( Q_7 \notin BQ(Q_6, B_2) \). Yet, BEChk then computes the fetch closure of \( X_{Q_6}^B \) over \( U_{Q_6} \) and sets \( BA(Q_6, B_2) \) to \( \{ B, C, F, E \} \). It then finds that all nontrivial attributes of \( R \) are in \( BA(Q_6, B_2) \) and hence updates \( BR(Q_6, B_2) \) to \( \{ R \} \). Consequently, it sets \( BQ(Q_6, B_2) \) to \( Q_7 \) as well. In the second iteration, BEChk builds an updated universal relation \( U_{Q_6} = \{ A, B, C, E, F, W \} \) since \( Q_7 \in BQ(Q_6, B_2) \) and \( \Sigma_{Q_6} \vdash \text{sum}(y) = E \). It continues to update \( BA(Q_6, B_2) \) to \( \{ B, C, E, F, W \} \), \( BR(Q_6, B_2) \) to \( \{ R \} \), and \( BQ(Q_6, B_2) \) to \( \{ Q_7, Q_6 \} \). It terminates after the third iterations and returns “Yes” for \( Q_6 \) under \( B_2 \).

Correctness & Complexity. To see that BEChk correctly checks the effective syntax \( L_B \) of Fig. 1, observe the following.

(1) For any fixed \( BQ(Q, B) \), the corresponding \( BA(Q, B) \) decided by rules \( \gamma_1, \cdots, \gamma_4 \) is exactly the fetch closure \( W^B \) (recall the definition of \( W^B \) above).

(2) The while loop propagates changes from BA to BR and to BQ, and finally to BA again, until reaching a fixed point w.r.t. the rules of Fig. 1.

BEChk can be implemented in \( O(p_Q(||Q|||B| + ||Q||)) \) time, where \( p_Q \) is the number of sub-queries in \( Q, ||Q|| \) is the number of relation atoms in \( Q, ||B|| \) is the number of attributes and aggregate fields in the relation atoms and predicates of \( Q, |B| \) and \( |B| \) are the number and total length of bag constraints in \( B \), respectively (see Table 1). Indeed, computing the fetch closure can be implemented in \( O(||Q|||B|| + ||Q||) \) time; and hence each while iteration is in \( O(||Q|||B|| + ||Q||) \) time; there are at most \( p_Q \) iterations.

Algorithm BEChk provides a constructive proof for property (c) of the effective syntax \( L_B \) in Theorem 7, i.e., it is in PTIME to check whether \( Q \in L_B \) for an \( ARD \) query \( Q \).

This also completes the proof of Theorem 7.

5.3 Generating bounded plans

We next provide an algorithm underlying component \( C3 \) of BEAS, denoted by BPlan. Given a bag access schema \( B \) and an \( ARD \) query \( Q \) that is determined to be boundedly evaluable under \( B \) by BEChk of Section 5.2, BPlan generates a bounded \( ARD \) query plan for \( Q \) under \( B \).

Algorithm BPlan. Given a boundedly evaluable \( ARD \) query \( Q \in L_B \) (see Section 4), BPlan generates a bounded plan \( \xi_Q \) for \( Q \) under \( B \) as follows: (1) fetch a bounded amount of data for each relation \( R \) that appears in \( Q \), and (2) carry out operations of \( Q \) over fetched data. While step (2) is straightforward, step (1) is rather involved.

To carry out step (1), BPlan generates bounded logical access plans (bLPAs). A bLP \( \xi_R \) for a relation \( R \) in \( Q \) fetches all values (partial tuples) of \( R \) that are necessary for evaluating \( Q \) with \( B \); moreover, \( \xi_R \) is a bounded \( ARD \) plan under \( B \). Intuitively, bLPAs play the same role as conventional DBMS access paths. But instead of accessing complete tuples by scan or index, bLPAs fetches values (partial tuples) using \( B \) such that the amount of data accessed is bounded.

More specifically, we give an algorithm, denoted by BAP, as a sub-procedure of BPlan to find a bLP \( \xi_R \) for \( R \) under \( B \). While there may exist exponentially many such bLPs, BAP aims at computing those with minimum cost.
After BAP computes bLAP $\xi_R$ for every relation $R$ in $Q$, algorithm BPlan generates a bounded plan $\xi_Q$ for $Q$ under $\mathcal{B}$, by replacing each $R$ in $Q$ with its bLAP $\xi_R$, and by carrying out RA$_{agg}$ operations of $Q$ on the data retrieved by $\xi_Q$.

In the rest of the section, we focus on algorithm BAP.

**Parametric cost measures.** To evaluate the quality of bLAPs found by BAP, we start with a generic class of cost functions. Conventional access path measures assess the cost of physical table-access methods, e.g., sequential scan and index scan$^{[1]}$. These metrics do not apply to bLAPs, which involve, e.g., fetch and joins. Hence, BAP employs a generic cost function $c(\xi_R)$ that takes user specified functions as parameters, to express various cost measures over $\xi_R$ as bLAPs, e.g., output size, data access, etc.

The cost of a bLAP $\xi_R$ for $R$ under $\mathcal{B}$, denoted by $c(\xi_R)$, is inductively defined in Table 2, with five user configurable function parameters $I_{\text{bs}}, I_{\text{b}}, I_{\text{cb}}, I_{\text{fetch}}$, and $I_{\text{gpBy}}$.

| bLAP $\xi_R$ | $c(\xi_R)$ of $\xi_R$ |
|-------------|-----------------|
| $\{c\}$    | 1               |
| $\sigma_c(\xi')$ | $c(\xi') \times \lambda_0(C)$ |
| $\xi_1 \bowtie \xi_2$ | $I_{\text{bs}}(c(\xi_1), c(\xi_2)) \times \lambda_0(C)$ |
| $\xi_1 \setminus \xi_2$ | $I_{\text{b}}(c(\xi_1), c(\xi_2))$ |
| $\pi_1(\xi')$ | $c(\xi')$ |
| $\text{gpBy}(\xi', X; \text{ragg}(V))$ | $I_{\text{gpBy}}(c(\xi'), \lambda_0(X))$ |
| $\text{fetch}(\xi', \varphi)$ with $\varphi = R[X \rightarrow Y, N]$ | $I_{\text{fetch}}(c(\xi'), N)$ |

By parameterizing these user configurable functions, we can support various measures for bLAPs. For example, to estimate the worst-case output size of $\xi$, we simply set (i) $I_{\text{bs}}(c_1, c_2)$ to $c_1 \times c_2$, (ii) $I_{\text{fetch}}(c', N)$ to $c' \times N$, (iii) $I_{\text{b}}(c_1, c_2)$ to $c_1 + c_2$, (iv) $I_{\text{gpBy}}(c', c) \times c$ to $c'$ if $c \neq 0$ and to 1 otherwise (assume $\lambda_0(X) = 0$ when $X = \emptyset$).

**Algorithm BAP.** Algorithm BAP works in two steps:

1. it reduces bLAPs to proofs of $R \in \mathcal{BR}(Q, \mathcal{B})$ and encodes all proofs with a directed graph $G(Q, \mathcal{B})$ in PTIME; and

2. it searches $G(Q, \mathcal{B})$ to find proofs with minimum cost, where a proof corresponds to a subgraph in the search trace.

Here a proof of $R \in \mathcal{BR}(Q, \mathcal{B})$ is a sequence of applica-

---

4Following query optimizer in DBMS, $\lambda_0(C)$, $\lambda_{\text{col}}(C)$, $\lambda_0(X)$ are coefficients that can be estimated from database statistics as a priori.

---

Below we outline BAP (see Appendix for its pseudo-code).

(1) **Reduction.** It reduces the problem of generating bLAPs for $R$ under $\mathcal{B}$ to finding proofs of $R \in \mathcal{BR}(Q, \mathcal{B})$. It encodes all proofs of $R \in \mathcal{BR}(Q, \mathcal{B})$ (hence all bLAPs for $R$) in a weighted directed graph $G(Q, \mathcal{B})$, where nodes encode (a) attributes $R[X]$ and $R[XY]$ in constraints $R[X \rightarrow Y, N] \in \mathcal{B}$, and (b) relations and sub-queries of $Q$. Edges encode value propagation among them. It ensures that each proof of $R \in \mathcal{BR}(Q, \mathcal{B})$ is encoded by a traversal from a dummy node $u_B$ to node $u_R$ encoding $R$ in $G(Q, \mathcal{B})$. Graph $G(Q, \mathcal{B})$ has at most $2|B| + |Q|$ nodes and $|B||(B| + |Q|)$ edges.

We illustrate reduced graphs with the following example, and defer the construction details to Appendix.

**Example 9.** Recall RA$_{agg}$ queries $Q_2$ of Example 2 and $Q_6$ of Example 5, and bag access schemas $B_1$ and $B_2$ from Example 4. Graphs $G(Q_2, B_1)$ and $G(Q_6, B_2)$ are shown in Fig. 3. Here $u_B$ is a dummy node connected to all constant attributes in $Q$. Edges with numeric weights are to encode deduction steps with rule $\gamma_1$ of Fig. 1, where the weights are the cardinality $N$s of the corresponding access constraints.

---

As will be show below, proofs of a relation $R \in \mathcal{BR}(Q, \mathcal{B})$ can be encoded as traversals from $u_B$ to $u_R$ in $G(Q, \mathcal{B})$.

(2) **Conditional Dijkstra search.** Algorithm BAP then adopts a Dijkstra-like search over $G(Q, \mathcal{B})$, from the dummy node $u_B$ to the relation node $u_R$ encoding $R$, such that the trace of the search encodes a bLAP (i.e., proof) for $R$ under $\mathcal{B}$.

It extends Dijkstra algorithm$^{[2]}$ as follows.

(a) Conditional expansion. Denote by $U_u$ the attribute, relation or sub-query encoded by a node $u$ in $G(Q, \mathcal{B})$. Note that $U_u$ may be deduced from attributes or
sub-queries encoded by multiple predecessors of $u$ as pre-
conditions in the proof of $R \in BR(Q, B)$. To capture this,
BAP visits a new node $u$ under the condition that $U_v$ can
be obtained from predecessors of $u$ via, e.g., joins or fetch.

With the condition, BAP ensures that for node $u$ en-
coding relation $R \in BR(Q, B)$, a traversal in $G(Q, B)$ from
$u_0$ to $u$ encodes a bLAP for relation atom $R$ in query $Q$
under $B$. To illustrate this, let us consider Example 10.

Example 10. A proof of $f \in BR(Q_2, B_1)$ (here $f$ de-
notes friend) consists of the deduction steps (1), (2) and
(4) in Example 7. It is encoded by the unique path from
$u_0$ to $u_f$ in $G(Q_2, B_1)$; similarly for the proof of$c \in BR(Q_2, B_1)$.

A more informative example is $S \in BR(Q_6, B_2)$. Its
proof is also described in Example 7. The proof is en-
coded by the traversal from $u_0$ to $u_S$, which is $G(Q_6, B_2)$
without the edge $(u_s, u_{Q_6})$ as shown in Fig. 3. Note that
although there are two simple paths from $u_0$ to $u_S$, none of
them is a valid traversal because of the conditional ex-
pansion.

The bLAPs encoded by these proofs are exactly sub-
plans of $Q_2$ and $Q_6$ given in Example 5. Indeed, the
bLAP $\xi$ for relation $c$ of $Q_2$ under $B_1$ is $(T_1, T_2)$ of
the bounded plan $\xi_{Q_2}$ in Example 5, and the bLAP $\xi_f$ for $f$ is
simply $T_1$. Similarly, the bLAP $\xi_R$ for $R$ of $Q_6$ under $B_2$ is
simply $T_1$ of $\xi_{Q_6}$ in Example 5; and bLAP $\xi_S$ for $S$ is $(T_1,
T_2, T_3)$ of $\xi_{Q_6}$.

Note that a bLAP may involve multiple relations via
fetch and join, e.g., $\xi_S$ for $S$ of $Q_6$. Hence its costs
cannot be assessed by traditional access path measures since
those methods are developed for evaluating the access
method of a single relation via, e.g., sequential or index
scan.

(b) Search revision. Note that the output of an RAse-
gry sub-query can be used to fetch attributes that have
already been deduced, possibly with a smaller cost re-
duced by $c(\xi_K)$. To retain the optimality of the search,
when visiting a node $u$ that encodes a sub-query of $Q$,
algorithm BAP checks whether this yields a better bLAP
by starting a new search from $u$ and marking all nodes as
unvisited. It terminates if it cannot further improve the
previously searched bLAPs.

Example 11. Continuing with Example 9, assume that we use $c(\xi_K)$ to express the worst-case output size of $\xi_K$ (recall its parameter functions described earlier). Then BAP computes exactly the bLAPs for $Q_2$ and $Q_6$ under $B_1$ and $B_2$, respectively, as described in Example 10. In particular, $c(\xi_1) = 5000$ and $c(\xi_2) = 5000 \times 193$ for $Q_2$
under $B_1$; $c(\xi_R) = 10$ and $c(\xi_S) = 10$ for $Q_6$ under $B_2$. In
this case, when computing $\xi_K$ for relation $S$ of $Q_6$
under $B_2$, it restarts the search once due to the aggregate sub-
query $Q_7$, which does not improve the bLAPs $\xi_R$ and $\xi_S$.

Correctness & Complexity. The correctness of al-
gorithm BAP is warranted by the following: (1) each
search trace of BAP encodes a proof of $R \in BR(Q, B)$;
and (2) a proof of $R \in BR(Q, B)$ encodes a bLAP for $R$
under $B$. BAP can be implemented in $O(|Q| \cdot |B| + |Q| \cdot \log(|Q| + 2|B|))$-time (ignoring the
complexity of parameter functions of $c(\xi_K)$). One can
verify that BAP restarts at most $N$ times, where $N$ is the
number of nodes in $G(Q, B)$.

Optimality. Algorithm BAP is able to find optimal
bLAPs for a large class of parameter functions for $c(\xi_K)$.
We defer detailed proofs of this optimality to Appendix.

6 Experimental study

We have developed BEAS@PG by extending PostgreSQL
with bounded evaluation. Using a benchmark and
two real-life datasets, we conducted four sets of experi-
ments to evaluate (1) the overall performance of
BEAS@PG vs PostgreSQL; and the effectiveness of
bounded evaluation for (2) bounded queries and (3)
unbounded queries.

Experimental setting. We start with the setting.

Benchmark. We used TPCh benchmark[25]. It uses
TPCH dbgen to generate 8 relations with 61 attributes of
different scales. It contains 22 built-in benchmark queries.

Real-life datasets. We also used two real-life datasets.

(a) US Air carriers (AIRCA) records flight and statis-
tical data of US air carriers. It consists of Flight On-Time
Performance Data[25] for departure and arrival informa-
tion, and Carrier Statistic data[26] for airline market
and segment data of the air carriers. It has 3 tables, 200
attributes, and about 16 GB of data with records from 1990
to 1997.

(b) UK MOT data (UKMOT) integrates the an-
onymised data[27] that records MOT tests and outcomes,
and the roadside survey of vehicle observations[28] that in-
cludes vehicles passing observation points in the UK. It
has 3 tables with 42 attributes, about 16 GB of data from
2007 to 2011.

Queries. To test the impact of query structures on the
effectiveness of bounded evaluation, we designed a gener-
ator to generate queries with different structures over the
two real-life datasets. More specifically, we manually cre-
ated 30 query templates for each of the two datasets
(Q1–Q15 are boundedly evaluable and Q16–Q30 are un-
bound), with 0 to 4 joins. The generator populates these
templates by randomly instantiating parameters in
the templates with values from the datasets, yielding 150
queries for each real-life dataset.

Access schema. We built access schemas with 59, 18
and 14 access constraints over TPCH, AIRCA and UK-
MOT, respectively. We extended TANE[26], an algorithm
for discovering functional dependencies, to first find can-
didate constraints $\varphi = R(X \rightarrow Y, N)$ on small sample
data datasets of 100 MB, and ranked them by their cardinalities
$N$’s. We then checked whether their $N$’s are insensitive
to the size of datasets $D$, by varying the size of $D$,
e.g., 200 MB and 500 MB. We picked those access con-
strants with small and size-insensitive \( N \)'s, such that the total size of the indices is at most 3 times of the size of its \( D \).

**Configuration.** For DBMS, we used PostgreSQL 9.6 with all optimization enabled (BEAS@PG is built with PostgreSQL 9.6). In favor of PostgreSQL, besides indices for access constraints, we also built the following extra indices for PostgreSQL: (1) for each access constraint \( R[\{X \rightarrow Y, N\}] \), we built a B-tree index on attributes \( X \) over \( R \) as well; (2) we built all primary key and foreign key indices; and (3) we also built B-tree on numerical attributes. Note that these were only for PostgreSQL, not built for BEAS@PG. We set the cost measure parameters of BEAS@PG as the worst-case output size estimation (recall Section 5.3).

The experiments were conducted on an Amazon EC2 Dense-storage instance m4.xlarge, with 16 GB of memory, 4 Intel Xeon E5-2676 vCPUs, and 500 GB of EBS SSD storage. Both the plan generation time and the execution time of the generated plans are included in evaluation time. All the experiments were run 3 times. The average is reported here.

**Experimental results.** We next report our findings.

**Exp-1: Overall performance.** We first report the evaluation time of 22 TPCH queries over 16 GB of TPCH data, and the 60 query templates over the entire AIRCA and UKMOT datasets, where evaluation time of a query template is the average of the evaluation time of its 5 instantiated queries.

(1) **Index size.** The indices of all the access constraints over TPCH, AIRCA and UKMOT account for 2.98, 0.01 and 0.25 times of the size of the datasets, respectively; the additional indices built only for PostgreSQL (in favor of the conventional DBMS) are of size 2.21, 0.87 and 1.5 times of that of TPCH, AIRCA and UKMOT, respectively.

(2) **Query overview.** None of the TPCH queries is boundedly evaluable under the access constraints selected. This is because the TPCH data generator scales cardinalities \( N \)'s of almost all candidate access constraints \( R[\{X \rightarrow Y, N\}] \) due to its simple scaling up strategy. This rules out most of the candidate constraints when we scale up to larger datasets while using a fixed threshold for \( N \). For the 60 query templates over AIRCA and UKMOT, 30 of them are boundedly evaluable under the access constraints used, 15 for each dataset. Note that one could build more access constraints to allow more bounded queries. We will evaluate the performance of BEAS@PG for bounded and unbounded queries in more details in Exp-2 and Exp-3, respectively.

(3) **Performance.** The results for TPCH, AIRCA and UKMOT are reported in Tables 3–5, respectively.

(a) BEAS@PG outperforms PostgreSQL on each and every query on all the three datasets, when all indices are enabled for PostgreSQL. It is \( 1.11 \times 10^4 \) times faster on average.

(b) Even though all TPCH queries are unbounded, over 16 GB of TPCH data, BEAS@PG is up to 40.46 times faster than PostgreSQL, and is on average 7.32 times faster. For unbounded queries over AIRCA and UKMOT, BEAS@PG is on average \( 1.32 \times 10^3 \) and \( 4.61 \times 10^3 \) times faster than PostgreSQL, respectively, up to \( 1.48 \times 10^4 \) and \( 6.10 \times 10^4 \) times.

(c) For bounded queries, BEAS@PG is \( 1.79 \times 10^4 \) and \( 3.66 \times 10^4 \) times faster than PostgreSQL on AIRCA and UKMOT, respectively, up to \( 3.44 \times 10^5 \) and \( 2.52 \times 10^5 \) times.

The results show that with a modest number (and size) of access constraints, BEAS@PG can speed up PostgreSQL on both bounded queries and unbounded queries, when all relevant indices are enabled for PostgreSQL, including those of access constraints and additional indices tailored for PostgreSQL. This verifies the effectiveness of bounded evaluation for generic queries, bounded or not, while the speedup is much larger for bounded queries, as expected.

Below we report more in-depth evaluation results for BEAS@PG versus PostgreSQL (with additional indices) for bounded queries (Exp-2) and unbounded queries (Exp-3).

**Exp-2: Effectiveness for bounded queries.** We next evaluated the impact of datasets \( D \) and queries \( Q \) on the evaluation time of BEAS@PG and PostgreSQL (with indices enabled), when queries \( Q \) are boundedly evaluable.

**Varying \( |D| \).** To evaluate the impact of \( |D| \), we partitioned AIRCA and UKMOT datasets by their date attributes (year and month), yielding subsets of sizes from 1 GB to 16 GB, consistent with how we scale up TPCH datasets when testing unbounded queries below in Exp-3. We did not use TPCH here since it has no boundedly evaluable queries. As shown in Figs. 4(a) and 4(b), (a) the evaluation time of BEAS@PG is indifferent to the size of \( D \), as expected for boundedly evaluable queries. (b) Bounded query plans work well with large \( D \). Indeed, BEAS@PG took less than 11.67 ms and 3.94 \( \times 10^2 \) ms for all queries over all subsets of AIRCA and UKMOT, respectively, no matter how large the datasets were. In contrast, even on the subsets of AIRCA and UKMOT of size 8 GB, PostgreSQL took \( 8.45 \times 10^4 \) ms and \( 3.88 \times 10^5 \) ms, respectively, up to \( 1.58 \times 10^7 \) ms and \( 7.80 \times 10^7 \) ms over the full datasets. That is, PostgreSQL is \( 1.35 \times 10^4 \) and \( 1.98 \times 10^5 \) slower than BEAS@PG on AIRCA and UKMOT, respectively, even with all relevant indices built. The larger the dataset is, the bigger the gap between PostgreSQL and BEAS@PG is for bounded queries.

**Varying \( Q \).** To evaluate the impact of queries \( Q \), we varied the complexity of \( Q \), measured as the number \#\( Q \) of joins in the query templates \( Q \), from 0 to 4, while using the entire AIRCA and UKMOT datasets. Note that for each query template, we instantiated 5 queries by set-
Table 3 TPCH query evaluation time on 16 GB (ms)

| Queries | Q1   | Q2   | Q3   | Q4   | Q5   | Q6   | Q7   | Q8   | Q9   | Q10  |
|----------|------|------|------|------|------|------|------|------|------|------|
| PostgreSQL | 8.16 × 10^3 | 3.68 × 10^3 | 1.02 × 10^3 | 1.56 × 10^3 | 1.50 × 10^3 | 2.07 × 10^3 | 9.53 × 10^3 | 3.71 × 10^3 | 5.03 × 10^3 | 2.28 × 10^4 | 7.05 × 10^4 |
| BEAS@PG  | 5.72 × 10^3 | 1.67 × 10^3 | 2.97 × 10^3 | 1.45 × 10^3 | 1.43 × 10^3 | 4.25 × 10^3 | 7.65 × 10^3 | 3.43 × 10^3 | 3.24 × 10^4 | 8.19 × 10^4 | 3.46 × 10^3 |
| Speedup  | 14.28 | 2.21 | 3.44 | 1.07 | 1.05 | 4.88 | 1.24 | 1.08 | 1.55 | 2.79 | 20.39 |

Table 4 Average query evaluation time on AIRCA (ms)

| Queries | Q1   | Q2   | Q3   | Q4   | Q5   | Q6   | Q7   | Q8   | Q9   | Q10  |
|----------|------|------|------|------|------|------|------|------|------|------|
| PostgreSQL | 7.00 × 10^3 | 7.23 × 10^3 | 2.06 × 10^3 | 9.04 × 10^4 | 4.44 × 10^4 | 9.41 × 10^4 | 1.36 × 10^5 | 1.44 × 10^5 | 9.41 × 10^4 | 1.38 × 10^5 |
| BEAS@PG  | 1.22 | 1.19 | 0.64 | 2.63 | 2.61 | 6.34 | 4.28 | 6.69 | 6.02 | 4.98 |
| Speedup  | 5.75 × 10^3 | 6.04 × 10^3 | 3.21 × 10^3 | 3.44 × 10^4 | 1.70 × 10^4 | 1.48 × 10^4 | 3.19 × 10^4 | 2.15 × 10^4 | 1.57 × 10^4 | 2.77 × 10^4 |
| Queries  | Q11  | Q12  | Q13  | Q14  | Q15  | Q16  | Q17  | Q18  | Q19  | Q20  |
| PostgreSQL | 1.46 × 10^4 | 9.46 × 10^4 | 1.49 × 10^5 | 1.56 × 10^5 | 10.63 × 10^4 | 4.46 × 10^4 | 4.65 × 10^4 | 4.22 × 10^4 | 4.25 × 10^4 | 9.29 × 10^4 |
| BEAS@PG  | 7.24 | 11.42 | 6.55 | 8.5  | 11.67 | 24.96 | 24.3 | 2.84 | 4.63 × 10^3 | 7.26 × 10^2 |
| Speedup  | 2.01 × 10^4 | 8.29 × 10^4 | 2.29 × 10^4 | 1.84 × 10^4 | 9.11 × 10^3 | 1.79 × 10^3 | 1.91 × 10^3 | 1.48 × 10^3 | 91.86 | 1.29 × 10^2 |
| Queries  | Q21  | Q22  | Q23  | Q24  | Q25  | Q26  | Q27  | Q28  | Q29  | Q30  |
| PostgreSQL | 9.30 × 10^4 | 8.98 × 10^4 | 9.44 × 10^4 | 9.41 × 10^4 | 1.39 × 10^5 | 1.47 × 10^5 | 9.58 × 10^4 | 1.75 × 10^5 | 1.85 × 10^5 | 1.31 × 10^5 |
| BEAS@PG  | 5.43 × 10^4 | 4.61 × 10^4 | 7.25 × 10^4 | 5.23 × 10^4 | 1.4 × 10^4 | 1.74 × 10^4 | 1.36 × 10^4 | 2.67 × 10^4 | 3.06 × 10^4 | 2.62 × 10^4 |
| Speedup  | 1.71 × 10^4 | 1.94 × 10^3 | 1.31 × 10^3 | 1.80 × 10^3 | 99.52 | 84.49 | 70.35 | 6.54 | 6.06 | 5.02 |

Ting its parameters with different values (hence these queries share the same query structure and #Q). The evaluation time of each query template is the average of all its instantiated queries.

The results are reported in Figs. 4(c) and 4(d). We find the following. (a) The complexity of Q has impacts on the performance of both BEAS@PG and PostgreSQL, as expected. They both take longer time for queries with more joins (i.e., #Q). However, (b) BEAS@PG scales much better with the number #Q of joins in Q than PostgreSQL (with indices). For instance, on average BEAS@PG found answers for all queries with #Q = 4 within 11.67 ms on full-sized AIRCA, while PostgreSQL takes 1.56 × 10^8 ms; that is, PostgreSQL is 1.34 × 10^8 times slower than BEAS@PG for large queries.

Remark: We find that when queries Q incur joins on keys only, PostgreSQL with extra key/foreign key indices built is almost as fast as BEAS@PG (e.g., TPCH Q4). However, as long as Q involves non-key attributes, e.g., many of the AIRCA and UKMOT queries, PostgreSQL performs poorly on big tables, even provided with all indices. Indeed, on average BEAS@PG outperforms PostgreSQL by 8.98 × 10^3 times and 1.76 × 10^4 times for all bounded queries over all subsets of AIRCA and UKMOT, respectively. The gap gets larger when the number of non-key attributes increases.

By looking into PostgreSQL’s plan and its EXPLAIN output, we find that this is partially due to the following reason. Given an access constraint R[X → Y, N], BEAS@PG fetches only distinct values of the relevant XY attributes, but PostgreSQL fetches entire tuples with irrelevant attributes of R, although those attributes are not needed for answering Q at all, no matter what indices are provided. This led to duplicated (X,Y) values when X is not a key, and the duplication got inflated rapidly by joins, e.g., XPLAIN output shows that PostgreSQL consistently accesses entire tables when there are non-key attributes.

Exp-3: Effectiveness for unbounded queries. In the same setting as in Exp-2, we evaluated the impact of D and Q on the performance of unbounded queries by BEAS@PG and PostgreSQL with indices enabled for PostgreSQL.

Varying #D, The results on AIRCA, UKMOT and TPCH are in Fig. 4(e), (f) and (g), respectively. Observe the following.
(a) BEAS@PG is able to speed up PostgreSQL even for queries that are not bounded under the available access constraints. On average, BEAS@PG is 7.22×10^2, 2.29×10^3 and 3.43 times faster than PostgreSQL for unbounded queries on AIRCA, UKMOT and TPCH, respectively. This is because while not all relations in these queries are bounded, bounded evaluation can still speed up their “bounded” subqueries, and hence remains faster than PostgreSQL.

(b) As opposed to evaluating bounded queries, both BEAS@PG and PostgreSQL are sensitive to the size of the datasets when evaluating unbounded queries. However, BEAS@PG scales much better than PostgreSQL, and their performance gap becomes larger when the dataset size increases. For example, when the dataset increases from 1GB to 16GB, the average processing time of BEAS@PG increases from 9.53×10^7 ms, 1.67×10^8 ms and 2.21×10^8 ms to 6.09×10^9 ms, 1.84×10^10 ms and 4.54×10^10 ms on AIRCA, UKMOT and TPCH, respectively. In contrast, PostgreSQL increases from 7.50×10^9 ms, 2.61×10^9 ms and 5.98×10^9 ms to 1.04×10^10 ms, 4.53×10^10 ms and 1.23×10^12 ms, respectively, even with all indices built and enabled.

Note that the speedup for unbounded TPCH queries is not as good as for AIRCA and UKMOT queries. This is because (i) the Ns of access constraints R(\{X \rightarrow Y, N\}) over TPCH scale linearly as the dataset gets larger, while those on AIRCA and UKMOT are more stable and independent of the dataset size; and (ii) joins in TPCH queries are mostly key/foreign key joins, and thus the extra key indices built for PostgreSQL can mimic bounded query plans used by BEAS@PG to some extent, reducing their performance gaps.

**Varying Q.** Varying the number \#Q of joins in the queries, the evaluation time of unbounded queries over AIRCA and UKMOT is reported in Figs. (4h) and (4i), respectively. The results tell us the following. (a) The processing time of BEAS@PG and PostgreSQL increases when the number of joins increases. However, (b) the gap between BEAS@PG and PostgreSQL becomes larger when \#Q increases from 0 to 4. For instance, over AIRCA, on average BEAS@PG and PostgreSQL take 17.37 ms and 4.43×10^3 ms, respectively, to answer queries with \#Q = 0; and the two take 2.78×10^5 ms and 1.64×10^5 ms, respectively, when \#Q = 4; the results over UKMOT are similar. Note that for bounded queries, the gap between the two is even larger (Exp-2).

**Summary.** We find the following. (1) BEAS@PG (PostgreSQL with BEAS built on top) does better than PostgreSQL for each and every query in all cases, even with extra indices built for the latter. On average BEAS@PG improves PostgreSQL by 7.32, 9.58×10^4 and 2.06×10^4 times for TPCH benchmark of 16GB, AIRCA and UKMOT, respectively, up to 40.46, 3.44×10^4, and 2.52×10^5 times in the best case. (2) For queries that are boundedly evaluable, BEAS@PG outperforms PostgreSQL by 1.9×10^4 and 3.6×10^4 times on AIRCA and UKMOT, respectively. (3) For queries with complicated joins, e.g., joins on non-key attributes (AIRCA and UKMOT queries), BEAS@PG is particularly effective, even for unbounded queries. For example, on average BEAS@PG improves PostgreSQL by 5.97×10^2 and 1.90×10^3 times for queries that are not boundedly evaluable over AIRCA and UKMOT, respectively. For cases where conventional DBMS does its best, e.g., table scan/aggregation and key-foreign key joins (most TPCH queries), BEAS@PG still does better than PostgreSQL. (4) The storage cost for indices of access schema is modest, accounting for 2.98, 0.01 and 0.25 times of the size of 16GB TPCH, AIRCA and UKMOT, respectively.

---

**Table 5: Average query template evaluation time on UKMOT (ms)**

| Queries | Q1  | Q2  | Q3  | Q4  | Q5  | Q6  | Q7  | Q8  | Q9  | Q10 |
|---------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| PostgreSQL | 1.37×10^3 | 6.47×10^2 | 7.80×10^3 | 4.62×10^2 | 4.65×10^3 | 3.86×10^3 | 5.8×10^2 | 3.91×10^2 | 5.72×10^3 | 5.99×10^3 |
| BEAS@PG | 0.55 | 2.73 | 0.62 | 16.13 | 75.2 | 5.62 | 1.50×10^2 | 3.78×10^2 | 54.79 | 1.89×10^2 |
| Speedup | 2.52 | 2.38 | 1.25 | 3.86×10^2 | 6.18×10^3 | 6.88×10^4 | 3.88×10^2 | 1.04×10^3 | 1.04×10^4 | 3.18×10^3 |

Querries Q21 Q22 Q23 Q24 Q25 Q26 Q27 Q28 Q29 Q30

Average (ms) 5.91 5.35 2.38 1.89 5.91 5.35 2.38 1.89 5.91 5.35

---
7 Related work

The related work is categorized as follows.

**Bounded evaluation.** The notion of bounded evaluation was introduced in [5], as an effort to formalize scale independence [19, 30, 31]. The latter aims to guarantee that a bounded amount of work is required to execute all queries in an application, regardless of the size of the underlying data. Under access schema proposed in [19], Fan et al. [5] defines boundedly evaluable RA queries. It establishes the complexity of deciding whether a query is boundedly evaluable, for queries in various fragments of RA, ranging from EXPSPACE-hard to undecidable. Bounded evaluation using views was studied in [32], focusing on its complexity bounds.

To cope with the undecidability of the bounded evaluability problem, an effective syntax was given for RA in [6] under the set semantics. Based on the syntax, algorithms were developed [6] for checking the bounded evaluability of RA queries \( Q \), and if affirmative, generating a bounded query plan for \( Q \). These issues were also studied in [33] for SPC, using a restricted form of query plans. Based on [6], a prototype BEAS for RA queries was developed [34].

This work extends the prior work in the following. (1) We define bag access schema, an extension of access schema of [5, 19] to support the bag semantics (Section 2). (2) We identify decidable special cases of the bounded evaluability problem that cover a variety of SQL queries commonly used in practice. (3) We develop an effective syntax for boundedly evaluable RA \texttt{aggr} queries under a bag access schema, supporting nested aggregations (Section 4). Moreover, the syntax allows us to make a larger class of RA queries bounded, improving the result of [6] for RA. (4) We extend BEAS [34] from RA to RA\texttt{aggr}, by seamlessly integrating bounded evaluation with DBMS query optimizers, which is quite different from [6, 34]. These extend DBMS with bounded evaluation, which was not studied in [5, 6, 33, 34].

**Query answering with constrained resources.** The objective of this work is to make big data analytics accessible to small companies under constrained resources. For queries that are not boundedly evaluable, an approach is to compute approximate answers under available re-
sources. Approximation techniques have been extensively studied, based on synopsis (e.g., [35–39]) or dynamic sampling (e.g., [40–42]). We have proposed a data-driven approximation scheme [17] that computes approximate answers $Q(D_Q)$ to an RA$_{agg}$ query $Q$ in a dataset $D$, by identifying a fraction $D_Q$ of $D$ under an extension of the access schema of [5]. It ensures a deterministic accuracy bound $n$: (a) for each tuple $s \in Q(D_Q)$, there exists an exact answer $t$ that is within distance at most $n$ from $s$, and (b) for each exact answer $t \in Q(D)$, there exists $s \in Q(D_Q)$ within distance $n$ from $t$.

This work differs from [17] in that we focus on computing exact answers instead of approximation. The techniques are hence quite different. In particular, a bag access schema carries the multiplicities of tuples to deal with the bag semantics, as opposed to distance bounds in access templates of [17]. This said, this work and [17] are complementary to each other. On one hand, the methods of [17] can be used to compute approximate answers to unbounded queries under constrained resources. On the other hand, the techniques developed in this work can be incorporated into the methods of [17], to improve the accuracy of approximate answers by making use of DBMS optimizers and bounded sub-plans.

**Indices.** Hash-based or tree-based, DBMS indices are typically defined at the tuple level [11], to retrieve tuple IDs and fetch full tuples. In contrast, a bag constraint $R([X \rightarrow Y, N])$ offers a value-based index. Bounded plans fetch distinct partial tuples (Y-values) for each input X-value, and thus reduce duplicated and unnecessary attributes in tuples fetched by DBMS, i.e., reduce data access and intermediate relations. The redundancies get inflated rapidly with joins. Moreover, the cardinality constraints in a bag access schema allow us to determine whether data access is bounded.

Related to bag access schema is a notion of access patterns, which require a relation to be accessed only by providing certain combinations of attributes, e.g., [43–45]. As opposed to access patterns, a bag access schema offers cardinality constraints, tuple multiplicity and indices. Moreover, it is not required to cover all the attributes of a relation and hence, allows us to fetch partial tuples and reduce redundancy. Further, this work studies bounded evaluation of RA$_{agg}$ queries and its integration with DBMS, which were not considered in the prior work on query answering under access patterns.

**Query optimization.** There has been a host of work on query optimization in DBMS, including access path selection [46], join optimization [47, 48] and recently, machine learning methods [49–51]. These focus on access path cost models for, e.g., main-memory concurrent systems, heuristics for join [47] and group-by [48] re-ordering, learned indices [50, 51] or optimizers [49, 52–54]. Our algorithms and techniques are complementary to the prior work, to incorporate bounded evaluation into DBMS query optimization.

**8 Conclusions**

We have presented an approach to extending DBMS with bounded evaluation of SQL queries. The novelty of the work consists of (a) a notion of bag access schema to support the bag semantics of nested aggregations; (b) decidable special cases of the bounded evaluability of RA$_{agg}$ queries; (c) an effective syntax to characterize boundedly evaluable RA$_{agg}$ queries; and (d) a framework and its underlying algorithms for integrating bounded evaluation with DBMS. Our experimental study has verified that the approach is promising. Together with the approximation scheme of [17], we hope that this work provides small businesses with a capacity for querying big data under constrained resources.

One topic for future work is to develop algorithms for discovering bag access schemas by incorporating machine learning techniques. Another topic is to extend bounded evaluation of SQL queries to column-oriented DBMS.

**Appendix**

**Bounded plans under $B$ (Section 2)**

The set $\Xi_B$ of bounded query plans under a bag access scheme $B$ is inductively defined in Fig. 5.

**Details of Algorithm BPlan (Section 5.3)**

We provide (1) the construction of weighted directed graph $G(Q, B)$ for generating bLAPs, (2) algorithm BAP, and (3) justification for the optimality of bLAPs found by BAP.

**Constructing $G(Q, B)$.** Let $V$ and $B$ be the node set and edge set of $G(Q, B)$, respectively. They are constructed as follows: (1) for each access constraint $R([X \rightarrow Y, N]) \in B$, (a) include $u_{R[X]}$, $u_{R[Z]}$ in $V$, referred to as BA-node; (b) include $(u_{R[X]}, u_{R[Y]})$ in $E$, referred to as a fetch-edge; (2) for each relation $S$, include a node $u_S$ in $V$, referred to as a BR-node; (3) for each sub-query $Q_s$ of $Q$, include $u_{Q_s}$ in $V$ as a BQ-node and $v_{Z(Q_s)}$ in $V$ as a BA-node, where $Z(Q_s)$ includes the output attributes of $Q_s$; (4) for each constant attribute $A = c$ in $Q$, there is a BA-node $u_A$ in $V$, and an additional node $u_A$; (5) for any two BA-nodes $u_X$ and $u_Y$, if $u_X$ (resp. $u_Y$) is not the head (resp. tail) of a fetch-edge, then $(u_X, u_Y)$ is
an edge in \( E \) if there exist \( A \in X \) and \( B \in Y \) such that \( \Sigma(R) = A \cap B \); (6) \((u, u_A)\) is an edge in \( E \) for every constant attribute \( A \) in \( Q \); (7) for any BA-node \( u_X \) and BR-node \( u_q \), if \( u_X \) is the tail of a fetch edge and \( X \) contains all nontrivial attributes of \( R \), then \((u_X, u_R)\) is an edge in \( E \); and (8) for any BR-node \( u_R \) and BQ-node \( u_{Q,v} \), \((u_R, u_{Q,v})\) is an edge in \( E \) if \( R \) is a relation of \( Q \).

In graph \( (Q, B) \), fetch-edges carry cardinality \( N \)'s in their encoded access constraints \( R(\{ X \to Y, N \}) \in B \) as their weights; each edge \((u_A, u_A)\) for constant node \( u_A \) has weight 1; and the other edges have no weight. Graph \( (Q, B) \) has at most 2 \( |B| + |Q| \) nodes and \(|B| + |Q| \) edges.

**Encoding proofs using \( (Q, B) \).** Traversing a fetch-edge \((u_{R,X}, u_{R,X})\) encodes fetch \( h(\xi_{R[X]}, \varphi) \) for some \( \varphi = R(\{ X \to Y, N \}) \in B \) that fetches values for \( Y \), where \( \xi_X \) is the bLAP encoded by the traversal from \( u_B \) to \( u_{R,X} \); which retrieves values for \( R[X] \) necessary for answering \( Q \); a traversal from a set \( S \) of BA-nodes \( u_X, \ldots, u_X \) to a BA-node \( u_Y \) encodes a bLAP \( \xi_Y = \varphi(Y(\xi_X, \ldots, \xi_X)) \) for \( Y \), where \( \xi_X(i, [1, n]) \) is the bLAP encoded by the traversal from \( u_b \) to \( u_X \) that retrieves values for \( X_i \). Other cases are similar.

**Algorithm 2. BAP.**

**Input:** RAnest query \( Q \), bounded relation \( R \) of \( Q \), bag access schema \( B \) and cost function \( c() \).

**Output:** A bLAP \( \xi_B \) for \( R \) under \( B \).

1. construct graph \( (Q, B) \);
2. \( P_Q := \emptyset; S_{ct} := \emptyset; H := \emptyset; D_T[u] := 0; L_T[u] := 0; GATE[u] := \emptyset; \)
3. for each \( u \) in \( (Q, B) \) do
4. if \( u \neq u_B \) then \( D_T[u] := +\infty; L_T[u] := 0; GATE[u] := L[u]; \)
5. \( P_Q.push(u)_B; \) //Initialization
6. while \( P_Q \neq 0 \) do
7. \( u := P_Q.top(); /\; P_Q \) pops out \( u \) with minimum \( D_T[u] \) in \( P_Q \)
8. \( S_{ct} := S_{ct} \cup \{u\}; \)
9. for each neighbor \( v \) of \( u \) that is not in \( S_{ct} \) do
10. \( GATE[v] := GATE[v] \setminus L[u]; \)
11. if \( (u, v) \) is a fetch-edge and \( D_T[v] > \Gamma_{fetch}(D_T[u], w(u, u)) \) then
12. \( P_Q.push(v); \)
13. \( D_T[v] := \Gamma_{fetch}(D_T[u], w(u, v)); L_T[v] := \{u\} \)
14. else if \( v \) is a BR-node then
15. \( P_Q.push(v); D_T[v] := D_T[u]; L_T[v] := \{u\} \)
16. else if \( v \) is a BQ-node and \( GATE[v] = \emptyset \) then
17. \( P_Q.push(v); D_T[v] := c(\xi_v); L_T[v] := \emptyset; \Gamma_{fetch} \) is the plan for \( Q \) encoded by \( v \), composed from predecessors \( pre(v) \) of \( v \) following the structure of \( Q \)
18. else if \( GATE[v] = \emptyset \) then
19. \( P_Q.push(v); \)
20. \((D_T[v], L_T[v], S_{ct}) := SC(u, pre(v) \cap S_{ct}, v); \)
21. if \( L_T[v] \) contains BQ-node \( u_q \) and \( D_T[v] \) and \( u_q \notin H \) then
22. \( D_T[v] := d; S_{ct} := \{u_q, v\}; H := H \cup \{v\} // restart the search
23 return bLAP that is encoded by the traversal from \( u_B \) to \( u_R \) recorded in \( L_T \).

**Algorithm 2. BAP.** BAP is given as Algorithm 2. It uses (a) \( GATE[u] \) to record the condition for visiting \( u \), e.g., for the head \( u_X \) of a fetch-edge, \( GATE[u] = X \); (b) \( L[u] \) to denote the attributes or relations \( u \) encoded; (c) \( D_T[u] \) to denote the cost of the part of bLAP encoded by the search trace from \( u_B \) to \( u \); (d) \( L_T[u] \) to store the nodes to be visited before visiting \( u \); (e) a priority queue \( P_Q \) for nodes to explore; and (f) sets \( S_{ct} \) of visited nodes and \( H \) of nodes triggered restarts.

Algorithm BAP starts the search from \( u_B \). It first initializes data structures (lines 2–5). It then iteratively explores nodes in \( P_Q \) (lines 6–22). The search extends the Dijkstra search with conditional node expansion controlled by \( GATE[v] \) and types of the edges (lines 9–22), using \( c() \) to calculate the traversal cost. It restarts the search if the output of a sub-query is used as an input for a fetch (i.e., to visit the head of a fetch-edge; determined by \( SC \) given as Algorithm 3) with a reduced cost (lines 21–22). It returns bLAP encoded by the search trace if restarts cannot improve its cost (line 23).

**Algorithm 3. SC.**

**Input:** visited vertex \( u \) and vertex set \( S_{ct} \), and vertex \( v \) to be visited.

**Output:** \( D_T[v], L_T[v] \) and updated \( S_{ct} \).
1. if \( GATE[v] \neq \emptyset \) then \( D_T[v] := \lambda(D_T[u], w(u, v)); L_T[v] := \{u\}; \)
2. return \( D_T[v], L_T[v], S_{ct} \)
3. \( W := L_T[v]; H_v := \emptyset; \)
4. while \( W \neq 0 \) do
5. \( choose u' \in S_{ct} \cap pre(v) \) with minimum \( g_a(\{D_T[u']\} \cup \{u'\}); \)
6. \( W := W \setminus L[u']; H_v := H_v \cup \{u'\}; \)
7. if \( (u, v) \) is a fetch-edge and \( \lambda(D_T[u], w(u, v)) < g_a(\{D_T[u']\} \cup \{u'\}); \)
8. \( D_T[v] := \lambda(D_T[u], w(u, v)); L_T[v] := \{u\} \)
9. else \( D_T[v] := g_a(\{D_T[u']\} \cup \{u'\}); L_T[v] := H_v \)
10. if \( H_v \) contains \( u' \) then \( S_{ct} := \{u\}; \)
11. return \( D_T[v], L_T[v], S_{ct} \);

**Optimality of BAP.** We say that cost \( c(\xi_v) \) is regular if (a) all parameter functions are monotonically non-decreasing w.r.t. each of their arguments; and (b) \( \Gamma_u, \Gamma_L, \Gamma_{LBN}, \Gamma_{fetch} \) and \( \Gamma_{fetch} \) are commutative and associative.

Denote \( \max_{x \in R(\{ X \to Y, N \}) \in B} Y \) by \( \#B \), where \( Y \) is the
number of attributes in $Y$. Then we have the following.

**Proposition 11.** Under any access schema $B$, BAP finds optimal bLAPs with regular cost functions if $\#B \leq 1$.

**Proof sketch.** We discuss SPC $Q$ first, followed by $R_{A_{agg}}$.

(1) $Q$ is in $SPC$. We first prove the following lemmas.

(a) For each node $u \in S_D$, $D[u]$ is the shortest distance from $u_0$ to $u$; and for each unvisited node $v$, $D[v]$ is the shortest distance from $u_0$ when traversing nodes in $S_D$ only.

(b) For each $u \in S_D$, the plan encoded by the traversal from $u_0$ to $u$ is of cost $D[u]$ when $c()$ is regular and $\#B = 1$.

For if these hold, then when BAP terminates, the encoded plan from $u_0$ to $u_R$ is an LAP for $R$ with minimum $c()$.

Lemma (a) can be proved by induction on the number of nodes in $S_D$, along the same line as Dijkstra’s optimality proof, by observing that $S_C(v, S)$ always selects the subset $S_0 \subseteq S$ with minimum cost w.r.t. $c()$ for visiting $v$ when $\#B = 1$. Lemma (b) can be verified by the same induction with (i) the observation that $D[u]$ mimics $c()$ for the encoded bLAP from $u_0$ to $u$, and (ii) the correspondence between proofs (traversals in $G(Q, B)$) and bLAPs under $B$ in the proof of Theorem 7 (part of the proof of Lemma (II)).

(2) $Q$ is in $R_{A_{agg}}$. For $R_{A_{agg}}$ queries, we show that BAP preserves the optimality of SPC with the following lemmas:

(c) Every sub-query $Q_s$ of $Q$ can improve searched LAP once.

(d) The order of sub-queries to restart does not change the final bLAP (up to equivalence).

Both lemmas are proved by induction on the number of $\neg$, $\cup$ and gBy operations in $Q$, using the condition that $\Gamma_x$ is associative and commutative and $c()$ is monotonic.

One may expect BAP to find optimal bLAP for more cases while remaining in PTIME. This is, however, beyond reach.

**Proposition 12.** For each integer $k > 1$, it is NP-hard to decide, given any bLAP access schema $B$ with $\#B = k$, $R_{A_{agg}}$ query $Q \in L_B$, relation atom $R$ of $Q$ and number $R$, whether there exists a bLAP $\xi_R$ for $R$ with cost $c(\xi_r) \leq r$, even when $c(\xi_r)$ is regular.

**Proof.** We show the NP-hardness by reduction from vertex cover (VC), which is NP-complete\[^{[1]}\]. An instance of VC consists of a graph $G(V, E)$ and an integer $n$. Given $G$ and $n$ (in binary form), VC is to decide whether there exists a subset $S \subseteq V$ such that (1) $S$ covers $G$, i.e., for any edge $e \in E$, there exists $u \in S$ on $e$; and (2) $|S| \leq n$.

Given an instance $G(V = \{v_1, \ldots, v_p\}, E = \{e_1, \ldots, e_q\})$ and $n$ of VC, we construct a database schema $\mathcal{R}$, a bag access schema $\mathcal{B}$ over $\mathcal{R}$ with $\#B = k$ ($k \geq 2$ is an integer), $R_{A_{agg}}$ $Q$ over $\mathcal{R}$, a bounded relation $R_B$ in $Q$, a real number $R$, a regular cost function $c()$ such that there exists a bLAP $\xi$ for $R_B$ under $B$ with $c(\xi) \leq r$ if and only if $G$ has a cover $S$ with $|S| \leq n$. More specifically, the reduction is given as follows.

(1) Database schema $\mathcal{R}$ consists of 3 relation schemas $R(A_0, A), S(A_1, \ldots, A_q, I)$, and $T(F_1, \ldots, F_k)$. Here relation $R$ is to encode the vertices of $G$, $S$ is to represent edges of $G$, and $T$ will be used to make $B$ with $\#B = k$.

(2) Query $Q$ is defined as follows:

$$\pi_{S}[\{\sigma_{R_1[A_0]=1}(R_1) \land \cdots \land \sigma_{R_p[A_0]=p}(R_p) \land S\} \land \phi) \land \psi]$$

where the join condition is $R_i[A] = S[A]$ if $v_i \in V$ is an end point of $e_j \in E$ in $G$. Here relation atom $R_i(i \in \{1, p\})$ is a renaming of relation schema $R$. Intuitively, the join condition encodes the edge relation of $G$, and $\pi_{S}[\{\}]$ is to ensure that every attribute in $S$ is nontrivial.

(3) The bag access schema $B$ consists of 3 access constraints:

- $\varphi_B = R(\{A_0 \rightarrow A, 2\})$.
- $\varphi_S = S(\{\{A_1, \ldots, A_q\} \rightarrow I, 1\})$, and $\varphi_T = T(\emptyset \rightarrow \{F_1, \ldots, F_k\}, 1)$.

Here $\varphi_B$ is to fetch values of attribute $R[A]$ (i.e., vertices of $G$), and $\varphi_S$ is to fetch edges encoded by $S[\{\}]$ using $R[A_1, \ldots, A_q] (i.e.,, edges of $G$). Note that $\#B = k$.

(4) The bounded relation $R_B$ is set to be $R$. One can easily verify that $R$ is bounded under $B$.

(5) We set $r = 2^n$. Note that this is in PTIME since $n$ in the VC instance is in binary.

(6) Function $c()$ is instantiated as follows: (i) $\Gamma_x(c(\xi_1), c(\xi_2)) = c_1 \times c_2$ if $\xi_1 \neq \xi_2$ and is $\max(c(\xi_1), c(\xi_2))$ otherwise. (ii) $\Gamma_{fetch}(c, N) = c \times N$; and (iii) we simply set all other functions as constants. Note that $c()$ is regular.

We show that $G$ has a vertex cover $S$ of size at most $n$ if and only if $R$ has a bLAP $\xi$ with cost $c(\xi) \leq r$.

$\Rightarrow$ Assume that $G$ has a vertex cover $S$ with $|S| \leq n$. We construct a bLAP $\xi$ for $R$ under $B$ as follows: $\xi = \text{fetch}\{\max_{i=1}^n \xi(A_i), \varphi_S\}$, where $\xi(A_i) = \text{fetch}\{\{v_i\}, \varphi_R\}$ if $v_i \in S$ (when $v_i$ is covered by multiple vertices in $S$, we pick one of them randomly). By the construction of $Q$ and by that $S$ covers all edges of $G$, $\xi$ is a bLAP for $S$ under $B$. From $|S| \leq n$, we know that $c(\xi) = \Gamma_x(\max_{i=1}^n \xi(A_i)) \times 1 \leq 2^n = r$.

$\Leftarrow$ Assume that $R$ has a bLAP $\xi$ under $B$ with cost $c(\xi) \leq 2^n$. By the join condition of $Q$ and $\varphi_S$, there exist at most $n$ distinct fetch operations for fetching $S[\{A_1, \ldots, S[A_0]$, i.e., $S[A_1], \ldots, S[A_q]$ can be fetched from at most $n$ numbers 1, 2, $\ldots, p$ using $\varphi_R$. This gives us a cover $S$ of $G$ as follows: $S = \{v_i(i \in \{1, p\}) | \exists j \in \{1, q\}. \xi(A_j) = \text{fetch}\{\{v_i\}, \varphi_R\}\}$. By the construction of $Q$, $S$ is a cover of $G$ with $|S| \leq n$. $\square$
Acknowledgements

The authors are supported in part by Royal Society Wolfson Research Merit Award WRM/R1/180014, ERC 652976, EPSRC EP/M025268/1, Shenzhen Institute of Computing Sciences, and Beijing Advanced Innovation Center for Big Data and Brain Computing.

Open access

This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made.

The images or other third party material in this article are included in the article’s Creative Commons license, unless indicated otherwise in a credit line to the material. Material that is not included in the article’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

[1] C. H. Papadimitriou. Computational Complexity, Reading, USA: Addison-Wesley, 1994.
[2] S. Abiteboul, R. Hull, V. Vianu. Foundations of Databases, Boston, USA: Addison Wesley, 1995.
[3] R. Horak. Telecommunications and Data Communications Handbook, New York, USA: Wiley, 2007.
[4] W. F. Fan, X. Wang, Y. H. Wu, D. Deng. Distributed graph simulation: Impossibility and possibility. Proceedings of the VLDB Endowment, vol.7, no.12, pp.1083–1094, 2014. DOI: 10.14778/2732977.2732983.
[5] W. F. Fan, F. Geerts, Y. Cao, T. Deng, P. Lu. Querying big data by accessing small data. In Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, ACM, Melbourne, Victoria, Australia, pp. 173–184, 2015. DOI: 10.1145/2745754.2745771.
[6] Y. Cao, W. F. Fan. An effective syntax for bounded relational queries. In Proceedings of 2016 International Conference on Management of Data, ACM, San Francisco, USA, 2016. DOI: 10.1145/2882903.2882942.
[7] The University of Edinburgh. Huawei deal to advance expertise in data science, [Online]. Available: https://www.ed.ac.uk/news/2017/huawei-deal-to-advance-expertise-in-data-science, June 14, 2017.
[8] Facebook. Introducing graph search beta, [Online]. Available: https://about.fb.com/news/2013/01/introducing-graph-search-beta/, January 15, 2013.
[9] I. Grujic, S. Bogdanovic-Dinic, L. Stoimenov. Collecting and analyzing data from e-government Facebook pages. In ICT Innovations, Ohrid, Macedonia, pp. 86–96, 2014.
[10] Facebook. Newsroom, [Online]. Available: http://newsroom.fb.com.
[11] R. Ramakrishnan, J. Gehrke. Database Management Systems, 2nd ed., New York, USA: McGraw-Hill Education, 2000.
[12] J. D. Ullman. Principles of Database Systems, 2nd ed., Computer Science Press, 1982.
[13] A. P. Stolboushkin, M. A. Taitslin. Finite queries do not have effective syntax. In Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM, San Jose, USA, pp. 277–285, 1995. DOI: 10.1145/212433.212477.
[14] A. van Gelder, R. W. Topor. Safety and translation of relational calculus queries. ACM Transactions on Database Systems, vol.16, no.2, pp.235–278, 1991. DOI: 10.1145/114325.105712.
[15] TPC. TPC-H, [Online]. Available: http://www.tpc.org/tpch/.
[16] W. F. Fan. Making Big Data Small, UK: British Royal Society, 2019. DOI: 10.1098/rspla.2019.0034.
[17] Y. Cao, W. F. Fan. Data driven approximation with bounded resources. Proceedings of the VLDB Endowment, vol.10, no.9, pp.973–984, 2017. DOI: 10.14778/3099622.3099628.
[18] Y. Cao, W. F. Fan, T. F. Yuan. Block as a value for SQL over NoSQL. Proceedings of the VLDB Endowment, vol.12, no.10, pp.1153–1166, 2019. DOI: 10.14778/3339490.3339498.
[19] W. F. Fan, F. Geerts, L. Libkin. On scale independence for querying big data. In Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM, Snowbird, USA, 2014. DOI: 10.1145/2594538.2594551.
[20] D. Abadi, P. A. Boncz, S. Harizopoulos, S. Ideos, S. Madden. The design and implementation of modern column-oriented database systems. Foundations and Trends® in Databases, vol.5, no.3, pp.197–280, 2013. DOI: 10.1561/1900000024.
[21] Microsoft SQL server columnstore indexes: Overview, [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview?view=sql-server-ver15.
[22] TPC. TPC-DS, [Online]. Available: http://www.tpc.org/tpcds/.
[23] M. R. Garey, D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, San Francisco, USA: W. H. Freeman, 1979.
[24] M. L. Fredman, R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM, vol.34, no.3, pp.596–615, 1987. DOI: 10.1145/28869.28874.
[25] Bureau of Transportation Statistics. The carrier on-time performance database, [Online]. Available: http://www.
transtats.bts.gov/DatabaseInfo.asp?DB_ID=120.

[26] Bureau of Transportation Statistics. The air carrier statistics database. [Online]. Available: http://www.transtats.bts.gov/DatabaseInfo.asp?DB_ID=110.

[27] Department for Transport. Anonymised mot tests and results. [Online]. Available: http://data.gov.uk/dataset/anonymised_mot_test, January 11, 2019.

[28] Department for Transport. Roadside survey of vehicle observations. [Online]. Available: https://data.gov.uk/dataset/52e1e2ab-5687-489b-a4d8-b207cd5d6767/roadside-survey-of-vehicle-observations.

[29] Y. Huhtala, J. Kärkkäinen, P. Pørkka, H. Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. The Computer Journal, vol.42, no.2, pp.100–111, 1999. DOI: 10.1093/comjnl/42.2.100.

[30] M. Armbrust, A. Fox, D. A. Patterson, N. Lamhan, B. Trushkowsky, J. Trutna, H. Oh. Scads: Scale-independent storage for social computing applications. In Proceedings of the 4th Biennial Conference on Innovative Data Systems Research, Asilomar, USA, 2009.

[31] M. Armbrust, S. Tu, A. Fox, M. J. Franklin, D. A. Patterson, N. Lamhan, B. Trushkowsky, J. Trutna. PIQL: A performance insightful query language. In Proceedings of ACM SIGMOD International Conference on Management of Data, ACM, Indiana, USA, pp.1207–1210, 2010. DOI: 10.1145/1807167.1807320.

[32] Y. Cao, W. F. Fan, F. Geerts, P. Lu. Bounded query rewriting using views. In Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, ACM, San Francisco, USA, pp.107–119, 2016. DOI: 10.1145/2902251.2902294.

[33] Y. Cao, W. F. Fan, T. Y. Wo, W. Y. Yu. Bounded conjunctive queries. Proceedings of the VLDB Endowment, vol.7, no.12, pp.1231–1242, 2014. DOI: 10.14778/2732977.2732996.

[34] Y. Cao, W. F. Fan, Y. H. Wang, T. F. Yuan, Y. C. Li, L. Y. Chen. BEAS: Bounded evaluation of SQL queries. In Proceedings of ACM International Conference on Management of Data, ACM, Chicago, USA, pp.1667–1670, 2017. DOI: 10.1145/3035918.3058748.

[35] S. Acharya, P. B. Gibbons, V. Poosala. Congressional samples for approximate answering of group-by queries. In Proceedings of ACM SIGMOD International Conference on Management of Data, ACM, Dallas, Texas, USA, pp.487–498, 2000. DOI: 10.1145/342009.345450.

[36] Y. E. Ioannidis, V. Poosala. Histogram-based approximation of set-valued query-answers. In Proceedings of the 25th International Conference on Very Large Data Bases, Edinburgh, Scotland, UK, pp.174–185, 1999.

[37] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, T. Suel. Optimal histograms with quality guarantees. In Proceedings of the 24rd International Conference on Very Large Data Bases, New York City, USA, pp.275–286, 2009.

[38] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, K. Shim. Approximate query processing using wavelets. The VLDB Journal, vol.10, no.2–3, pp.199–223, 2001.

[39] G. Cormode, M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In Proceedings of the 31st International Conference on Very Large Data Bases, ACM, Trondheim, Norway, 2005.

[40] B. Babcock, S. Chaudhuri, G. Das. Dynamic sample selection for approximate query processing. In Proceedings of ACM SIGMOD International Conference on Management of Data, ACM, San Diego, USA, pp.539–550, 2003. DOI: 10.1145/827757.827822.

[41] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri, B. Ding. Quickr: Lazily approximating complex AdHoc queries in BigData clusters. In Proceedings of International Conference on Management of Data, ACM, San Francisco, USA, pp.631–646, 2016. DOI: 10.1145/2882903.2882940.

[42] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, I. Stoica. BlinkDB: Queries with bounded errors and bounded response times on very large data. In Proceedings of the 8th ACM European Conference on Computer Systems, ACM, Prague, Czech Republic, pp.29–42, 2013. DOI: 10.1145/2465351.2465355.

[43] C. Li. Computing complete answers to queries in the presence of limited access patterns. The VLDB Journal, vol.12, no.3, pp.211–227, 2003. DOI: 10.1007/s00778-002-0085-6.

[44] M. Benedikt, J. Leblay, B. ten Cate, E. Tsamoura. Generating Plans from Proofs: Synthesis Lectures on Data Management, vol.8, no.1, pp.1–205, 2016. DOI: 10.2200/S00703ED1V01Y201602DTM043.

[45] A. Bash, B. Ludäscher. Processing first-order queries under limited access patterns. In Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM, Paris, France, pp.307–318, 2004. DOI: 10.1145/1055558.1055601.

[46] M. S. Kester, M. Athanassoulis, S. Idreos. Access path selection in main-memory optimized data systems: Should I scan or should I probe? In Proceedings of ACM International Conference on Management of Data, ACM, Chicago, USA, pp.715–730, 2017. DOI: 10.1145/3035918.3064049.

[47] T. Neumann. Query simplification: Graceful degradation for join-order optimization. In Proceedings of ACM SIGMOD International Conference on Management of Data, ACM, Rhode Island, USA, pp.403–414, 2009. DOI: 10.1145/1559845.1559889.

[48] M. Eich, P. Fender, G. Moerkotte. Faster plan generation through consideration of functional dependencies and keys. Proceedings of the VLDB Endowment, vol.9, no.10, pp.756–767, 2016. DOI: 10.14778/2977797.2977802.

[49] B. L. Ding, S. Das, R. Marcus, W. T. Wu, S. Chaudhuri, V. R. Narasayya. AI meets AI: Leveraging query executions to improve index recommendations. In Proceedings of International Conference on Management of Data, ACM, Amsterdam, The Netherlands, pp.1241–1258, 2019. DOI: 10.1145/3299869.3324957.

[50] T. Kraska, A. Beutel, E. H. Chi, J. Dean, N. Polyzotis. The case for learned index structures. In Proceedings of International Conference on Management of Data, ACM,
Yang Cao received the B.Sc. degree from Beihang University, China. He received the Ph.D. degree from University of Edinburgh, UK. He is a faculty member in the School of Informatics, University of Edinburgh, UK. He is the recipient of SIGMOD Research Highlight award 2018, SIGMOD Best Paper award 2017, and Microsoft Research Asia Fellowship. His research has been invited to publish in TODS special issues on “Best of SIGMOD 2017” and “Best of PODS 2016”, and in the Computer Journal special issue on “Best of BICOD 2015”.

His research interests include query processing, graph data management and distributed databases.

E-mail: yang.cao@ed.ac.uk (Corresponding author)
ORCID iD: 0000-0001-7984-3219

Wen-Fei Fan received the B.Sc. degree and M.Sc. degree from Peking University, China. He received the Ph.D. degree from University of Pennsylvania, USA. He is the Chair of Web Data Management at the University of Edinburgh, UK, the Chief Scientist of Shenzhen Institute of Computing Science, and the Chief Scientist of Beijing Advanced Innovation Center for Big Data and Brain Computing, China. He is a fellow of the Royal Society (FRS), a Fellow of the Royal Society of Edinburgh (FRSE), a Member of the Academy of Europe (MAE), an ACM Fellow (FACM), and a foreign member of Chinese Academy of Sciences. He is a recipient of Royal Society Wolfson Research Merit Award in 2018, ERC Advanced Fellowship in 2015, the Roger Needham Award, UK in 2008, Yangtze River Scholar, China in 2007, the Outstanding Overseas Young Scholar Award, China in 2003, the Career Award, USA in 2001, and several Test-of-Time and Best Paper Awards USA (Alberto O. Mendelzon Test-of-Time Award of ACM PODS 2015 and 2010, Best Paper Awards for SIGMOD 2017, VLDB 2010, ICDE 2007 and Computer Networks 2002).

His research interests include database theory and systems, in particular big data, data quality, data sharing, distributed query processing, query languages, recommender systems and social media marketing.

E-mail: wenfei@inf.ed.ac.uk
ORCID iD: 0000-0001-5419-2656

Teng-Fei Yuan received the B.Eng. degree from Shandong University, China. He is Ph.D. degree candidate in LFCS, School of Informatics, University of Edinburgh, UK.

His research interest is development of BEAS, a system for bounded evaluation of SQL queries.

E-mail: tengfei.yuan@ed.ac.uk