Data Driven Approximation with Bounded Resources (Full Version)

Yang Cao Wenfei Fan

University of Edinburgh & Beihang University
{*yang.cao@, wenfei@inf*}*.ed.ac.uk*

Abstract

This paper proposes BEAS, a resource-bounded scheme for querying relations. It is parameterized with a resource ratio $\alpha \in (0, 1]$, indicating that given a big dataset D, we can only afford to access an α-fraction of D with limited resources. For a query Q posed on D, BEAS computes exact answers $Q(D)$ if doable and otherwise approximate answers, by accessing at most $\alpha|D|$ amount of data in the entire process. Underlying BEAS are (1) an access schema, which helps us identify and fetch the part of data needed to answer Q, (2) an accuracy measure to assess approximate answers in terms of their relevance and coverage *w.r.t.* exact answers, (3) an Approximability Theorem for the feasibility of resource-bounded approximation, and (4) algorithms for query evaluation with bounded resources. A unique feature of BEAS is its ability to answer unpredictable queries, aggregate or not, using bounded resources and assuring a deterministic accuracy lower bound. Using real-life and synthetic data, we empirically verify the effectiveness and efficiency of BEAS.

1. Introduction

It is costly to compute answers $Q(D)$ to a query Q in a big dataset D. It is NP-complete to decide whether a tuple is in $Q(D)$ even for SPC query Q (selection, projection, Cartesian product) [6]. It easily takes hours to join tables with millions of tuples [38]. One might think that parallelism would solve the problem. However, there are queries for which the running time cannot be substantially reduced when we add more processors [25]. Moreover, small businesses can often afford limited resources such as processors.

Is it feasible to query big data with bounded resources?

We tackle this challenge by proposing a resource-bounded scheme for approximate query answering. It takes a *resource ratio* $\alpha \in (0, 1]$ as a parameter, indicating that our available resources can only access an α-fraction of a big dataset D. Given α, D and a query Q over D, it identifies $D_Q \subseteq D$, and computes approximate answers, denoted by $Q(D_Q)$, and an accuracy lower bound $\eta \in (0, 1]$ such that

(1) $|D_Q| \leqslant \alpha|D|$, where $|D_Q|$ is the size of D_Q; and

(2) $\mathsf{accuracy}(Q(D_Q), Q, D) \geqslant \eta$.

Intuitively, it computes $Q(D_Q)$ by accessing at most $\alpha|D|$ tuples in the process. Thus it can scale with D when D grows big by setting α small. Moreover, $Q(D_Q)$ assures a *deterministic* lower bound η such that under query relaxation [15, 30], (a) for *each* tuple $s \in Q(D_Q)$, there exists an exact answer t that is within distance at most η from s, and (b) for *each* exact answer t, there exists $s \in Q(D_Q)$ within distance η from t. That is, $Q(D_Q)$ includes only "relevant" answers and "covers" all exact answers within distance η. It finds sensible answers in users' interest, and suffices for exploratory queries, *e.g.,* real-time problem diagnosis on logs [8].

The objective is ambitious. Nonetheless, it is feasible under an *access schema* $\mathcal{A}$, a set of access templates. An access template is a combination of a cardinality constraint and an index. It helps us control how data is fetched from a dataset.

Example 1: (1) Consider a database schema $\mathcal{R}_0$ with relations (a) person(pid, city, address), stating that pid lives at address in city, (b) friend(pid, fid), recording a friend fid of pid, and (c) poi(address, type, city, price), describing the type, price and city of POI. A query Q_1 is to *find me hotels that cost at most $95 per night and are in a city where one of my friends lives*, from Graph Search of Facebook [22]:

select h.address, h.price
from poi **as** h, friend **as** f, person **as** p
where f.pid = p_0 **and** f.fid = p.pid **and** p.city = h.city **and** h.type = "hotel" **and** h.price $\leqslant$ 95

where p_0 indicates "me". It is costly to compute $Q_1(D_0)$ in a "big" instance D_0 of $\mathcal{R}_0$, *e.g.,* Facebook has billions of users and trillions of friend links [26]. Is it possible to answer Q_1 in D_0 given a small α, *e.g.,* 10^{-4}, *i.e.,* when our available resources can afford to access at most $10^{-4} * |D_0|$ tuples?

This is doable by using an access schema. There are many choices of access schema. To illustrate the idea, below we use $\mathcal{A}_0$ consisting of the following access templates:

- ψ_1: poi({type, city} $\to$ {price, address}, 1, (e_p^1, e_a^1)),
  ...
- ψ_m: poi({type, city} $\to$ {price, address}, 2^m, (e_p^m, e_a^m)),
- φ_1: friend(pid $\to$ fid, 5000, 0),
- φ_2: person(pid $\to$ city, 1, 0),

Here $m = \lceil \log_2 M \rceil$ and M is the maximum number of distinct poi tuples in D_0 grouped by (type, city). For $i \in [1, m]$, ψ_i says that given any (type, city)-value (c_t, c_c), there exists a set T of at most 2^i (price, address) values in poi of D_0. Moreover, for each poi tuple (c'_a, c_t, c_c, c'_p) in D_0, there exists $(c_p, c_a) \in T$ such that the (price, address)-value (c'_p, c'_a) differs from (c_p, c_a) within distance (e_p^i, e_a^i). That is, T represents (price, address) values that correspond to (c_t, c_c) with at most 2^i tuples, subject to distances (e_p^i, e_a^i), for $i \leqslant \lceil \log_2 M \rceil$. In addition, an index is built on (type, city) for T to be efficiently fetched. Note that by using the index for ψ_m, we can fetch all exact (price, address) values from D_0, *i.e.,* with distance $(e_p^m, e_a^m) = (0, 0)$; this is why we set $m = \lceil \log_2 M \rceil$.

Similarly, φ_1 states that each pid has at most 5000 friends, a limit enforced by Facebook [26], and these fid's can be fetched by using an index for φ_1; and φ_2 says that each pid lives in 1 city, which can be fetched via an index for φ_2. The indices fetch exact values, as indicated by distance 0.

Assume $\alpha|D_0| > 10000$, as in Facebook datasets D_0. Then under $\mathcal{A}_0$, we can find hotels by accessing at most $\alpha|D_0|$ tuples as follows: (a) fetch a set T_1 of fid's with p_0 by accessing at most 5000 friend tuples using φ_1; (b) for each fid in T_1, fetch 1 associated city with φ_2, yielding a set T_2 of at most 5000 city values; (c) for each city c in T_2, fetch at most 2^{k_α} (price, address) pairs corresponding to ("hotel", c) by using ψ_{k_α}, where $k_\alpha = \lfloor \log_2(\alpha|D_0| - 10000) \rfloor$; and (d) return a set S of those (price, address) values with price at most $(95 + e_p^{k_\alpha})$, as approximate answers to Q_1 in D_0.

Note that the entire process accesses at most $5000 + 5000 + 2^k_\alpha \leqslant \alpha|D_0|$ tuples in total. Moreover, the set S of answers is accurate: (1) for each hotel $h_0(c_p, c_a)$ in the exact answers $Q(D_0)$, there exists (c'_p, c'_a) in S that are within distance $e_p^{k_\alpha}$ and $e_a^{k_\alpha}$ of c_p and c_a, respectively; and (2) for each hotel $h'(c'_p, c'_a)$ in S, its price c'_p exceeds 95 by at most $e_p^{k_\alpha}$, *e.g.*, $e_p^{k_\alpha} = 4$ and $c'_p = 99$, and c'_a is the address of hotel h'.

(2) Consider query Q_2 to *find the cities where my friends live*:

```
select  p.city
from    friend as f,  person as p
where   f.pid = p0 and f.fid = p.pid
```

Then steps (a) and (b) above compute $Q_2(D)$ by using φ_1 and φ_2 alone, and by accessing at most 10000 tuples no matter how big D_0 grows. That is, the resource-bounded scheme is able to compute exact answers for certain queries. □

Contributions. This paper proposes BEAS (Boundedly EvAluable Sql), a resource-bounded framework for querying big relations. Given an application, it finds an access schema $\mathcal{A}$ for its dataset D. Under a resource ratio α, to answer Q in D, it identifies an α-fraction D_Q by reasoning about the cardinality constraints in $\mathcal{A}$, and fetches D_Q by using the indices of $\mathcal{A}$. It computes approximate answers $Q(D_Q)$ that are assured above a deterministic accuracy lower bound.

(1) Access schema (Section 2). We extend the access schema of [11,23] with access templates to support approximate query answering with bounded resources. With the extension, we formalize query plans under a resource ratio α.

(2) Accuracy measure (Section 3). We propose an RC-measure for accuracy under the assumption that query relaxation [15, 30] is allowed. As opposed to prior accuracy metrics [17, 27, 28, 34], the RC measure assesses approximate answers in terms of both their relevance and coverage *w.r.t.* exact answers, and allows a deterministic accuracy lower bound for query answers computed by resource-bounded algorithms.

(3) BEAS: foundation and architecture (Section 4). We formalize the resource-bounded approximation scheme. We present the Approximability Theorem. It assures that for any database schema $\mathcal{R}$, there exists an access schema $\mathcal{A}$ such that for all instances D of $\mathcal{R}$ that conform to $\mathcal{A}$, all resource ratios $\alpha \in (0, 1]$ and all queries over D, aggregate or not, BEAS identifies $D_Q \subseteq D$ and an RC bound η such that $|D_Q| \leqslant \alpha|D|$ and $\mathsf{accuracy}(Q(D_Q), Q, D) \geqslant \eta$. Moreover, the larger resource ratio α is, the higher the accuracy η is. In addition, when possible under small α, it is able to compute exact answers (*i.e.*, $\eta = 1$).

We also show how BEAS can be readily built on top of commercial DBMS, such that with bounded resources, it computes exact answers $Q(D)$ when possible, and approximate answers $Q(D_Q)$ otherwise with a deterministic η.

(4) Algorithms (Sections 5–7). We show the Approximability Theorem in three steps. We start with a resource-bounded approximation algorithm for answering SPC queries (Section 5). We then extend the algorithm to RA (relational algebra, Section 6). We show how to support set difference (universal quantification) when the resource ratio forbids us to scan the entire dataset. Finally, we study RA extended with aggregate functions and group-by construct. We show that the theorem also holds on aggregate queries (Section 7).

(5) Empirical study (Section 8). Using real-life and synthetic data, we experimentally verify the effectiveness of BEAS. We find the following. (a) BEAS computes approximate answers with accuracy $\eta \geqslant 0.85$ for SPC, and $\eta \geqslant 0.82$ for RA, aggregate or not, when $\alpha \geqslant 5.5 \times 10^{-4}$. Better yet, η gets higher when α increases, or when D grows bigger under the same α. (b) BEAS is able to find *exact answers* when α is as small as 2.6×10^{-6} for SPC and 4.1×10^{-6} for RA on a dataset of 60GB. (c) BEAS is efficient, taking at most 10.2 seconds on datasets of 200 million tuples when α is 5.5×10^{-4}, as opposed to *more than 3 hours* by PostgreSQL and MySQL. (d) BEAS outperforms sampling [17], histograms [28] and BlinkDB [8] under RC measure and MAC measure [28]. When α is 1.5×10^{-4}, its RC (resp. MAC) accuracy is 24, 3.4 and 2.0 (resp. 18.25, 2.4 and 1.9) times better than the other three, respectively.

A unique feature of BEAS is its ability to (1) answer unpredictable and generic queries, (2) comply with any *resource ratio* α, and (3) guarantee a *deterministic* accuracy lower bound η. Queries are *unpredictable* if we assume no prior knowledge about work load, query predicates or the frequency of columns used for grouping and filtering (QCS) [8]. A query is *generic* if it is aggregate or not. The ability is promising for providing small businesses with a functionality of big data analysis, despite their limited resources.

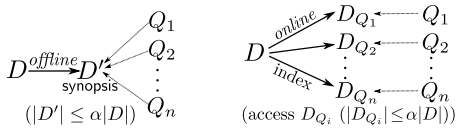
Related work. We classify related work as follows.

Bounded evaluation. This work is an extension of the study of bounded evaluation [11–13, 23, 24]. Under a set $\mathcal{A}$ of access constraints introduced in [24] a query Q is *boundedly evaluable* [23] if for all datasets D that satisfy $\mathcal{A}$, there exists $D_Q \subseteq D$ such that $Q(D) = Q(D_Q)$ and the time for identifying D_Q (and hence $|D_Q|$) is determined by $\mathcal{A}$ and Q, independent of $|D|$. The problem for deciding whether Q is boundedly evaluable was studied in the absence of $\mathcal{A}$ [24], under $\mathcal{A}$ [23], and with views [12]. The problem is shown undecidable for Q in RA and EXPSPACE -hard for Q in SPC [23]. In light of the complexity, an effective syntax was developed in [11] to characterize boundedly evaluable RA queries, analogous to the study of (undecidable) safe relational calculus queries. Algorithms were provided to check the bounded evaluability, and if so, generate bounded query plans to compute exact answers, for SPC [13] and RA queries [11].

BEAS computes exact answers of boundedly evaluable queries by using the algorithms of [11] for checking bounded evaluability and generating bounded query plans. In addition, it supports resource-bounded approximate query answering, and differs from the prior work in the following.

(1) Access schema. The access constraints of [11–13, 23, 24] are a special case of access templates with distance 0 (see Section 2). All queries on a dataset can be "covered" by access templates and approximately answered with bounded resources (Approximability Theorem). In contrast, not all queries are boundedly evaluable under access constraints.

(2) Approximate query answer. BEAS takes a resource ratio



(a) *One-size-fit-all* (b) *Dynamic reduction*

Figure 1: Data reduction schemes

α as a parameter, and computes approximate answers to *all* queries under α with an accuracy bound, aggregate or not. In contrast, bounded evaluation is not parameterized by α; it handles only RA queries that are boundedly evaluable.

(3) Techniques. Resource-bounded approximation is quite different from bounded evaluation. BEAS uses the chase [6] to generate query plan, enforces set difference without scanning the entire data, and handles aggregate functions and group-by. These issues were not studied in [11–13, 23, 24].

Approximation. Approximate query answering is typically based on either (a) synopsis [7, 14, 16, 28, 29], or (b) dynamic sampling (*e.g.,* [8, 9, 20, 31]). The former is to compute a synopsis D' of a dataset D, and use D' to answer all queries posed on D. Closer to our work is BlinkDB [8]. Assuming predictable QCSs, *i.e.,* "the frequency of columns used for grouping and filtering does not change over time", BlinkDB selects samples from historical QCS patterns, and caches them as views. It answers restricted aggregate queries using the samples instead of D, and offers probabilistic error rates.

Our scheme radically differs from the prior work as follows.

(1) Queries. Prior approaches "substantially limit the types of queries" [8]; they often target aggregate (on a single attribute) queries over a star-schema with key-foreign-key joins, and assume that workload, query predicates or group-by attributes are known in advance [8, 18, 20]. In contrast, BEAS works on unpredictable queries, aggregate or not.

(2) Accuracy. Prior methods give either no accuracy guarantee, or estimate probabilistic "error bars" for approximation of a single aggregate attribute [8, 20, 31]. In contrast, we assure *deterministic accuracy* on each approximate answer.

(3) Techniques. As shown in Fig. 1, BEAS adopts *dynamic data reduction* that identifies D_Q for each input query Q via *bounded* data access, as opposed to an *one-size-fit-all* synopsis D' [7, 14, 16, 28, 29]. While [8, 20, 31] use dynamic sampling, [20, 31] do not support bounded data access; [8] employs stratified sampling; given an error bar on the value distribution of an aggregate attribute, [20] builds samples accordingly. As remarked above, all these target restricted aggregate queries, and warrant no deterministic accuracy. Moreover, they require the entire set of exact values on group-by attributes, which may not be doable under small α.

Access patterns. Related is also query answering under access patterns, which require a relation to be accessed only by providing certain combinations of attributes [10, 33, 35]. This work differs from the prior work as follows. (a) Unlike access patterns, an access template imposes both cardinality constraints and "restricted" data accesses via indices. It is not required to cover all the attributes of a relation and hence, allows us to fetch partial tuples and reduce redundancy (see Section 2). (b) We target approximate query answering under a resource ratio α, and guarantee to answer all queries under access templates. Hence the results and techniques of BEAS are quite different from those for access patterns.

Resource-bounded algorithms. Related to this work are also anytime algorithms [41] and bounded-cost planning [40] for budgeted search, and sublinear-time algorithms [37] that read only part of the input. In contrast to the prior work, (1) we offer accuracy bounds, while [40,41] do not; (2) our algorithms are deterministic, while almost all sublinear-time algorithms are randomized or probabilistic [37]; and (3) we conduct search guided by access schema to maximize accuracy bound, a technique different from the prior work.

2. Access Schema and Bounded Query Plans

We start with access schema and bounded query plans.

2.1 Access Schema for Approximation

A database schema $\mathcal{R}$ is a collection $(R_1, \ldots, R_n)$ of relation names. Each $R(A_1, \ldots, A_h)$ in $\mathcal{R}$ has attributes A_i of domain U_i for $i \in [1, h]$. We assume a function $\mathsf{dis}_{A_i}\colon U_i \times U_i \to \mathbb{R}$ to measure the distance between two A_i-attribute values, where $\mathbb{R}$ denotes real numbers. As usual, we assume that the function dis_{A_i} satisfies the triangle inequality.

For instance, for $\mathsf{poi}(\mathsf{address}, \mathsf{type}, \mathsf{city}, \mathsf{price})$ of Example 1, $\mathsf{dis}_{\mathsf{price}}(95, 99) = 99 - 95$ by subtraction; and $\mathsf{dis}_{\mathsf{address}}$ measures physical distance between two locations.

It is *not* necessary to define dis_A for each A. Its default is a *trivial* distance function, defined as $\mathsf{dis}_A(x, y) = +\infty$ if $x \neq y$ and $\mathsf{dis}_A(x, y) = 0$ otherwise, *e.g.,* for ID attributes.

Access schema. An *access template* over $\mathcal{R}$ has the form

$$\psi = R(X \to Y, N, \bar{d}_Y),$$

where R is a relation schema in $\mathcal{R}$, X and Y are sets of attributes of R, N is a positive integer, and $\bar{d}_Y$, referred to as the *resolution tuple* of ψ, is a tuple of attributes Y with real numbers, *i.e.,* for each $B \in Y$, $\bar{d}_Y[B]$ is a value of $\mathbb{R}$.

Let $D_Y(X = \bar{a}) = \{t[Y] \mid t \in D, t[X] = \bar{a}\}$; *i.e.,* it is the set of all Y-values corresponding to X-value $\bar{a}$ in D.

An instance D of R *conforms to* ψ, denoted by $D \models \psi$, if there exists an index on D such that given any X-value $\bar{a}$,

- it accesses and returns a set $\tilde{D}_Y^N(X = \bar{a})$ of at most N distinct tuples in $D_Y(X = \bar{a})$, and
- for each $t \in D_Y(X = \bar{a})$, there exists $t' \in \tilde{D}_Y^N$ such that $\mathsf{dis}_{\mathcal{A}}(t[B], t'[B]) \leqslant \bar{d}_Y[B]$ for each attribute $B \in Y$,

Intuitively, ψ enforces a constraint: for each X-value $\bar{a}$, there is a sample $\tilde{D}_Y^N$ of no more than N distinct tuples that represents $D_Y(X = \bar{a})$ and satisfies resolution $\bar{d}_Y$. Moreover, the sample can be efficiently fetched via the index.

The access constraints of [11,24] are a special case of access template when $\bar{d}_Y = \bar{0}$, *i.e.,* $\tilde{D}_Y^N(X = \bar{a}) = D_Y(X = \bar{a})$. It is to fetch exact values $D_Y(X = \bar{a})$ corresponding to $\bar{a}$.

For instance, φ_1, φ_2 and ψ_i's of Example 1 are access templates, while φ_1 and φ_2 are access constraints of [11, 24].

An *access schema* $\mathcal{A}$ *over* $\mathcal{R}$ is a set of access templates over $\mathcal{R}$. An instance D of $\mathcal{R}$ *conforms to* $\mathcal{A}$, denoted by $D \models \mathcal{A}$, if D conforms each access template in $\mathcal{A}$.

2.2 Query Plans under Access Schema

We use access schema $\mathcal{A}$ over $\mathcal{R}$ to control accesses to instances of $\mathcal{R}$ and comply to resource ratio α. We formalize the idea in terms of query plans under $\mathcal{A}$. Consider an RA query Q over $\mathcal{R}$ with selection σ, projection π, Cartesian product $\times$, union $\cup$, set difference $-$ and renaming ρ.

Bounded query plans. Following [11], *A plan ξ under $\mathcal{A}$* is an index-only plan that uses indices of $\mathcal{A}$ to access data in a quantified manner. More specifically, ξ has the form:

$$\xi : T_1 = \delta_1, \ldots, T_n = \delta_n,$$

where for each $i \in [1, n]$, δ_i is one of the following:

(a) a set of constants;

(b) a relational operation on T_j's for $j < i$, *e.g.,* $T_j \setminus T_k$ for $j < i$ and $k < i$; or

(c) $\mathsf{fetch}(X \in T_j, R, Y, \varphi)$, where $j < i$, and φ is either an access constraint $R(X \to Y, N, 0)$ in $\mathcal{A}$, or an access template $R(X \to Y, N, \bar{d}_Y(k))$ in $\mathcal{A}$; the fetch operation retrieves a set W from D, where W is $D_{XY}(X = \bar{a})$ if φ is an access constraint, and is $\tilde{D}^k_Y(X = \bar{a})$ if φ is a template; it returns $\bigcup_{\bar{a} \in T_j(D)} \{(\bar{a}, \bar{b}) \mid \bar{b} \in W\}$.

Intuitively, ξ executes its operations one by one on a dataset D [11], each computing a relation to be used by subsequent operators. Its result $\xi(D)$ is the relation of the last operation. It controls data access in a *quantified manner* via fetch, using indices in access templates of $\mathcal{A}$.

A *query plan for Q under $\mathcal{A}$* is a plan ξ such that (a) all constants in ξ are from Q, and (b) for all instances $D \models \mathcal{A}$ of $\mathcal{R}$, $\xi(D) = Q(D)$ if for each fetch operation of ξ with access template ψ, the resolution $\bar{d}$ of ψ is upgraded to $\bar{0}$.

That is, when ξ were allowed to fetch data without "errors" (when N in ψ is large), it would compute exact answers $Q(D)$. Bounded plans for Q under $\mathcal{A}$ are defined solely based on their output and do not have to share the structure of Q.

For a resource ratio $\alpha \in (0, 1]$, a plan ξ for Q under $\mathcal{A}$ is *α-bounded in D* if it accesses at most $\alpha|D|$ tuples in D, where the number of tuples accessed by ξ can be deduced from constants N's in the templates used in ξ.

For instance, Example 1 gives an α-bounded plan for Q_1 in D_0. It picks template ψ_{k_α} based on α. The larger α is, the smaller $e_p^{k_\alpha}$ and $e_a^{k_\alpha}$ are, and the more accurate S is. When $k_\alpha = \lceil \log_2 M \rceil$, the plan computes exact $Q_1(D_0)$. Intuitively, we control the amount of data fetched by constants N in $\mathcal{A}$, and retrieve the data using the indices of $\mathcal{A}$.

Bounded evaluation of [11–13, 23, 24] can be modeled as a special case of bounded query plans under access schema $\mathcal{A}$. A query Q is *boundedly evaluable under $\mathcal{A}$* if it has a query plan ξ under $\mathcal{A}$ that uses access constraints only. For any $D \models \mathcal{A}$, ξ computes exact answers $Q(D)$.

For instance, Q_2 of Example 1 is boundedly evaluable.

3. Accuracy Measure

To assess the quality of a set S of approximate answers to a query Q in dataset D, we propose an accuracy measure, assuming that query relaxation is allowed to explore sensible answers. The new measure is characterized by two functions:

- coverage distance $\delta_{\mathsf{cov}}(Q, S, t)$ to measure how well an exact answer $t \in Q(D)$ is "covered" by S; and
- relevance distance $\delta_{\mathsf{rel}}(Q, D, s)$ to measure how relevant each approximate answer $s \in S$ is to Q in D.

With these, we define $\mathsf{accuracy}(S, Q, D)$, referred to as *the* RC*-measure* of S to Q in D, in terms of *coverage ratio* $\mathsf{F}_{\mathsf{cov}}(S, Q, D)$ and *relevance ratio* $\mathsf{F}_{\mathsf{rel}}(S, Q, D)$ as follows:

$$\mathsf{F}_{\mathsf{cov}}(S, Q, D) = \frac{1}{1+\max_{t \in Q(D)} \delta_{\mathsf{cov}}(Q, S, t)},$$
$$\mathsf{F}_{\mathsf{rel}}(S, Q, D) = \frac{1}{1+\max_{s \in S} \delta_{\mathsf{rel}}(Q, D, s)},$$
$$\mathsf{accuracy}(S, Q, D) = \min(\mathsf{F}_{\mathsf{rel}}(S, Q, D), \mathsf{F}_{\mathsf{cov}}(S, Q, D)).$$

In particular, (1) when $Q(D) = \varnothing$, we define $\mathsf{F}_{\mathsf{cov}}(S, Q, D) = 1$ for any S, empty or not; and (2) when $S = \varnothing$, we let $\mathsf{F}_{\mathsf{cov}}(S, Q, D) = 0$ if $Q(D) \neq \varnothing$ and hence $\mathsf{accuracy}(S, Q, D) = 0$.

Below we define the distance functions. We start with RA queries under set semantics (Section 3.1). We then extend to aggregate queries, coping with bag semantics (Section 3.2).

3.1 Distance Functions for RA Queries

We define *distance* $d(t, t') = \Sigma_{A \in \mathsf{R}} |\mathsf{dis}_A(t[A], t'[A])|$ for tuples t and t' of relation R, the sum of attribute differences.

Coverage. For a tuple $t \in Q(D)$, we define its *coverage distance* $\delta_{\mathsf{cov}}(Q, S, t)$ *to approximate answers S for* RA Q as

$$\min_{s \in S} d(s, t).$$

It assesses how well S covers an exact answer $t \in Q(D)$.

Relevance. One might want to define $\delta_{\mathsf{rel}}(Q, D, s)$ as $\min_{t \in Q(D)} d(s, t)$, the minimum distance between s and exact answers in $Q(D)$. This, however, denies sensible answers that are overlooked by Q, *e.g.,* hotel of \$99 per night for Q_1 of Example 1. To rectify this, we use relaxed query Q_r instead of Q, along the same lines as query relaxation [15, 30].

The *relaxed query* Q_r of Q with a positive number r is derived from Q such that all selections $\sigma_{A=c}$ and $\sigma_{A=B}$ in Q are replaced with $\sigma_{|\mathsf{dis}_A(A,c)| \leqslant r}$ and $\sigma_{|\mathsf{dis}_A(A,B)| \leqslant 2r}$, respectively.

Intuitively, Q_r relaxes selection conditions in Q to explore sensible answers. Bound r controls the quality of $Q_r(D)$ *w.r.t.* Q: (a) $\sigma_{|\mathsf{dis}_A(A,c)| \leqslant r}$ ensures that the A-attribute values in $Q_r(D)$ are within distance r to constant c in Q; and (b) $\sigma_{|\mathsf{dis}_A(A,B)| \leqslant 2r}$ allows both A and B attributes to be relaxed by r, and hence their difference is bounded by $2r$.

The *relevance distance* $\delta_{\mathsf{rel}}(Q, D, s)$ *of s to Q in D* is

$$\min_{r \geqslant 0} \max(r, \min_{t \in Q_r(D)} d(s, t)).$$

That is, $\delta_{\mathsf{rel}}(Q, D, s)$ measures how relevant s is as an answer to Q in D, by striking a balance between (a) relaxation range r, and (b) the distance of s to the closest ones in $Q_r(D)$. By taking the minimum of the two, it determines range r, *i.e.,* how far selections in Q should be relaxed.

When query relaxation is not allowed, we can fix $r = 0$ in $\delta_{\mathsf{rel}}(Q, D, s)$, and the RC-measure remains intact.

Observe the following: (1) $\mathsf{accuracy}(S, Q, D)$ is *deterministic*: each answer $s \in S$ is warranted relevant. (2) Both $\mathsf{F}_{\mathsf{rel}}(S, Q, D)$ and $\mathsf{F}_{\mathsf{cov}}(S, Q, D)$ are in the range $(0, 1]$. The larger they are, the more accurate S is. (3) Both $\mathsf{F}_{\mathsf{rel}}(S, Q, D)$ and $\mathsf{F}_{\mathsf{cov}}(S, Q, D)$ are 1 if $S = Q(D)$, the exact answers.

Justification. Several accuracy measures have been studied for approximation, classified as (a) counting-based, *e.g.,* F-measure and regret ratio [34]; (b) distance-based, *e.g.,* Hausdorff distance [27] and MAC [28], using distance functions to measure either the gap between approximate and exact answers [28], or the "losslessness" of a synopsis; and (c) confidence probabilities for aggregate queries, *e.g.,* BlinkDB [8] and sampling-based synopses [7, 17]. However, they do not work well on resource-bounded approximation for generic queries, or do not give a deterministic accuracy.

Example 2: Recall query Q_1 and access schema $\mathcal{A}_0$ from Example 1. Consider our familiar F-measure $F(S, Q, D) = 2 \frac{\mathsf{precs}(S,Q,D) \ \mathsf{recall}(S,Q,D)}{\mathsf{precs}(S,Q,D)+\mathsf{recall}(S,Q,D)}$, where $\mathsf{precs}(S, Q, D) = \frac{|S \cap Q(D)|}{|S|}$ and $\mathsf{recall}(S, Q, D) = \frac{|S \cap Q(D)|}{|Q(D)|}$. Even with the indices of $\mathcal{A}_0$, any deterministic algorithms that access only $\alpha|D|$ tuples may return approximate answers S with F-measure $F(S, Q, D) = 0$. This happens when S includes no hotels of price at most \$95. That is, by the F-measure, the approximation answers (algorithms) are no "good" for Q_1 *w.r.t.* α.

In contrast, S may include sensible answers, *e.g.,* hotels of \$99 per night, even by looking up the indices of $\mathcal{A}_0$ alone. This is reflected by its RC-measure with a non-0 value. □

In light of these, we propose to use relevance to measure how well approximate answers S serve users' need. The RC-measure is bi-criteria, assessing both the *relevance* of S and its *coverage* of exact answers $Q(D)$. As opposed to measures of type (a) above, it assures a non-zero accuracy bound for nonempty S so that we can compare the quality of different approximations. Its coverage subsumes metrics of type (b), and its relevance explains why a tuple is in S. Unlike (c), it works on generic queries beyond aggregates and offers deterministic accuracy, instead of probabilistic error rates.

3.2 Distance Functions for Group-By Aggregation

We now extend the distance functions to aggregate queries. We consider $\mathsf{RA}_{\mathsf{aggr}}$ [21], an extension of RA with a group-by construct. It has the form $Q = \mathsf{gpBy}(Q', X, \mathsf{agg}(V))$, where (a) Q' is an RA query, (b) X is a set of attributes in the output schema $R_{Q'}$ of Q', (c) V is an attribute in $R_{Q'}$, and (d) agg is one of max, min, avg, sum or count. The output of Q is a relation of schema R_Q, consisting of $\mathsf{agg}(V)$ and attributes in X. Written in SQL, Q is as follows:

select X, $\mathsf{agg}(V)$
from $R_1, \ldots, R_l$
where C **group by** X

Distances for $\mathsf{RA}_{\mathsf{aggr}}$. Consider an $\mathsf{RA}_{\mathsf{aggr}}$ query Q.

(1) Q is $\mathsf{gpBy}(Q', X, \mathsf{agg}(V))$, when agg *is* min *or* max .

- $\delta_{\mathsf{rel}}(Q, D, s) = \delta_{\mathsf{rel}}(Q', D, s)$ if there is no $s' \in S$ such that $s \neq s'$ and $s[X] = s'[X]$, and it is $+\infty$ otherwise.
- $\delta_{\mathsf{cov}}(Q, S, t) = \delta_{\mathsf{cov}}(Q', S, t)$.

Intuitively, (a) the condition for $\delta_{\mathsf{rel}}(Q, D, s)$ enforces the group-by semantics, *i.e.,* there exist no duplicated X-values in S; and (b) for min and max, $\delta_{\mathsf{rel}}(Q, D, s)$ and $\delta_{\mathsf{cov}}(Q, S, t)$ inherit $\delta_{\mathsf{rel}}(Q', D, s)$ and $\delta_{\mathsf{cov}}(Q', S, t)$, respectively, since approximate answer s to Q in D is also an approximate answer to Q' in D. Observe that the semantics of min and max is enforced by the coverage distance on attribute V.

(2) Q is $\mathsf{gpBy}(Q', X, \mathsf{agg}(V))$ when agg *is* avg, count *or* sum.

- $\delta_{\mathsf{rel}}(Q, D, s) = \delta_{\mathsf{rel}}(\pi_X(Q'), D, s[X])$ if there is no $s' \in S$ with $s \neq s'$ and $s[X] = s'[X]$, and it is $+\infty$ otherwise.
- $\delta_{\mathsf{cov}}(Q, S, t) = \mathsf{min}_{s \in S} d_{\mathsf{agg}}(s, t)$, where $d_{\mathsf{agg}}(s, t) = \Sigma_{A \in X} |\mathsf{dis}_A(s[A], t[A])| + f_{\mathsf{agg}}(t[V], s[V])$.

Here $f_{\mathsf{agg}}()$ is a distance function on the aggregate values, *e.g.,* $f_{\mathsf{agg}}(v, v') = |v - v'|$ as commonly found in practice.

In contrast to case (1) above, for avg, count and sum, aggregate values $t[V]$ and $s[V]$ may not be in the active domain of D. Hence $\delta_{\mathsf{rel}}(Q, D, s)$ is determined by $\delta_{\mathsf{rel}}(\pi_X(Q'), D, s[X])$, which measures how "qualified" s is to Q in D. For coverage, we measure how well S covers t via an extended coverage distance $\delta_{\mathsf{cov}}(Q', S, t)$ with f_{agg} on the aggregated attribute V, which enforces the semantics of aggregation.

Example 3: Consider an $\mathsf{RA}_{\mathsf{aggr}}$ query $Q_1' = \mathsf{gpBy}(Q_1, \mathsf{h.city}, \mathsf{count}(\mathsf{h.address}))$, to *find the numbers of hotels specified in Q_1 (Example 1) grouped by* city, replacing h.price with h.city in the **select** clause of Q_1. Assume for instance that $Q_1'(D)$ is $\{t_1 = (\mathrm{NYC}, 100), t_2 = (\mathrm{Chicago}, 140)\}$, and S is $\{s_1 = (\mathrm{NYC}, 80), s_2 = (\mathrm{Chicago}, 150)\}$. Then for $i \in [1, 2]$, $\delta_{\mathsf{rel}}(Q_1', D, s_i) = \delta_{\mathsf{rel}}(\pi_{\mathsf{h.city}}(Q_1), D, s_i[\mathsf{city}]) = 0$, *i.e., s_i* is relevant to Q_1'; and $\delta_{\mathsf{cov}}(Q_1', D, t_1) = 0 + |100 - 80| = 20$, $\delta_{\mathsf{cov}}(Q_1', D, t_2) = 0 + |140 - 150| = 10$, measuring the difference in count. □

symbols	notations
$\mathcal{R}$, R	database schema $\mathcal{R}$ and $R \in \mathcal{R}$
α, η	resource ratio and accuracy bound
ψ, $\mathcal{A}$	access template, access schema
ξ	query plan with fetch
$\delta_{\mathsf{rel}}(Q, s)$, $\delta_{\mathsf{cov}}(Q, t, s)$	distance functions (relevance, coverage)
$\mathsf{accuracy}(S, Q, D)$	the RC-measure of S *w.r.t.* $Q(D)$

Table 1: Notations used in the paper

The notations of this paper are summarized in Table 1.

4. Resource Bounded Query Answering

We next introduce the resource-bounded approximation scheme (Section 4.1) and the framework BEAS (Section 4.2).

4.1 Resource Bounded Approximation

A *resource-bounded approximation scheme* under an access schema $\mathcal{A}$ is an algorithm $\Gamma_{\mathcal{A}}$ such that for any $\mathsf{RA}_{\mathsf{aggr}}$ query Q (relational algebra extended with aggregates), *any* resource ratio $\alpha \in (0, 1]$ and *any* dataset $D \models \mathcal{A}$, it generates (a) an α-bounded query plan ξ_α for Q under $\mathcal{A}$ in D and (b) a bound $\eta \in [0, 1]$ such that the RC-measure $\mathsf{accuracy}(\xi_\alpha(D), Q, D) \geqslant \eta$, assuming that query relaxation is allowed.

The approximation scheme has the following properties.

(1) It takes resource ratio α as a parameter, allowing us to query big D with bounded resources by setting α small.

(2) Algorithm $\Gamma_{\mathcal{A}}$ computes plan ξ_α *without* accessing D. It takes only Q, $\mathcal{A}$ and budget $\alpha|D|$ as input. It generates plan ξ_α that computes $\xi_\alpha(D)$ by accessing at most $\alpha|D|$ tuples, and guarantees relevance and coverage of at least η.

(3) As shown in Fig. 1, $\Gamma_{\mathcal{A}}$ is based on *dynamic data reduction*: for each input query Q, it generates a (different) ξ_α guided by Q and $\mathcal{A}$, as opposed to one-size-fit-all synopsis.

Approximability. We now justify the feasibility.

Theorem 1 [Approximability Theorem]*: (1) For any dataset D, there exists an access schema $\mathcal{A}$ such that $D \models \mathcal{A}$.*

(2) Under $\mathcal{A}$, there exists an approximation scheme $\Gamma_{\mathcal{A}}$ such that for any ratio $\alpha \in (0, 1]$ and any $\mathsf{RA}_{\mathsf{aggr}}$ query Q, $\Gamma_{\mathcal{A}}$ computes an α-bounded plan ξ_α for Q in D and accuracy bound $\eta_\alpha \in [0, 1]$ such that when query relaxation is allowed,

- $\mathsf{accuracy}(\xi_\alpha(D), Q, D) \geqslant \eta_\alpha$*; and*
- *if $\alpha_1 \geqslant \alpha_2$, then $\eta_{\alpha_1} \geqslant \eta_{\alpha_2}$.* □

That is, on any database D, we can build access schema $\mathcal{A}$. For *any* resource ratio α and *all* $\mathsf{RA}_{\mathsf{aggr}}$ queries Q, $\mathcal{A}$ allows us to answer Q with an α-bounded plan ξ_α. The larger α we can afford, the better approximate answers $\xi_\alpha(D)$ we get.

As a proof of Theorem 1(1), we give a simple access schema $\mathcal{A}_t$. Consider the schema $\mathcal{R}$ of D. For a relation $R \in \mathcal{R}$, let D_R be the instance of R in D, $|D_R|$ be the number of tuples in D_R, $M_R = \lceil \log_2 |D_R| \rceil$, and $\mathsf{attr}(R)$ be the set of attributes of R. Then for each $R \in \mathcal{R}$, $\mathcal{A}_t$ includes access templates $\psi_k^R = R(\varnothing \to \mathsf{attr}(R), 2^k, \bar{d}_k)$ for all $k \in [0, 2^{M_R}]$.

Intuitively, $\mathcal{A}_t$ provides M_R levels of granularity for approximation of D_R, while $\psi_{M_R}^R$ is an access constraint with $\bar{d}_{M_R} = \bar{0}$. Note that the total size of indices for all the M_R templates is at most $2|D_R|$. Under $\mathcal{A}_t$, given a query Q and a ratio α, $\Gamma_{\mathcal{A}}$ generates a plan ξ_α and fine-tunes k in ψ_k^R for each D, to maximize the accuracy. We will provide resource-bounded approximation schemes for SPC, RA and $\mathsf{RA}_{\mathsf{aggr}}$ in Sections 5–7, respectively, as a proof of Theorem 1(2).

Choice of access schema. The simple $\mathcal{A}_t$ just aims to show the existence of resource-bounded approximation. As will be seen in Sections 5–7, over any access schema that subsumes $\mathcal{A}_t$, our approximation algorithms can find an α-bounded plans ξ_α for queries and deduce accuracy bound η.

In practice we extend $\mathcal{A}_t$ with more access templates and constraints, user-defined or discovered from D, which improve accuracy η. In other words, η deduced from $\mathcal{A}_t$ is a lower bound for the accuracy of approximate answers.

As suggested in [11], algorithms for discovering functional dependencies can be extended to mine access constraints. This method can be extended to discover access templates, with aggregates to compute cardinality bounds and sampling to pick representative tuples. Discovery of an optimal access schema is, however, harder. It aims to minimize response time, maximize the accuracy of query answering, and minimize the size of indices of the access schema. We defer the full treatment of the discovery problem to another paper.

Implementation. There are many approaches to constructing the indices of access schema, each with cons and pros. Below we give one that optimizes the resolutions (and thus optimizes the accuracy of bounded plans) based on K-D trees [19], and compare the approach with other approaches.

We first give details on how indices are built for access templates. We then describe how to organize indices for access templates and constraints together for an access schema.

We distinguish the following four cases.

(1) Indices for access templates in $\mathcal{A}_t$. For each relation $R \in \mathcal{R}$ and instance D_R of R, we can build indices on D_R for access templates $\psi_0^R, \ldots, \psi_{M_R}^R$ of $\mathcal{A}_t$ over R all together, by invoking a K-D tree construction once as follows.

(a) Build a K-D tree [19] T_D of D_R by treating tuples as m_R-dimensional points *w.r.t.* distance functions, where $m_R = |\mathsf{attr}(R)|$.

(b) For each $k \in [0, M_R]$, the index for ψ_k^R is a table $T_k^R(N, \mathsf{attr}(R))$ consisting of 2^k tuples such that $\pi_N(T_k^R) = \{2^k\}$, and $\pi_{\mathsf{attr}(R)}(T_k^R)$ consists of the 2^k tuples at the $(k+1)$-th level of T_D. For each attribute B of R,

$$\bar{d}_k[B] = \mathsf{max}_{t \in \pi_{\mathsf{attr}(R)}(T_k^R)} \mathsf{max}_{t_1, t_2 \in \mathsf{desc}(t)} |\mathsf{dis}_B(t_1[B], t_2[B])|,$$

i.e., the maximum error on attribute B introduced by representing D with the index for ψ_k, where $\mathsf{desc}(t)$ is the set of tuples that are descendants of t in T_D. Intuitively, $\bar{d}_k[B]$ is the maximum "width" on attribute B for tuples at the k-th level of T_D; it can be computed along with the construction of the K-D tree of D_R.

(2) Indices for access templates beyond $\mathcal{A}_t$. We next extend the construction to more general access templates $\psi_0 = R(X \to Y, 2^0, \bar{d}_0), \ldots, \psi_k = R(X \to Y, 2^k, \bar{d}_k)$. Their indices over a relation D_R of R can be constructed by extending the above as follows. Let D_X be $\pi_X(D_R)$, *i.e.,* the set of distinct X-values in D_R. For each $\bar{a} \in D_X$, let $D_Y(X = \bar{a})$ be the set of Y-values in D_R associated with $\bar{a}$, *i.e.,* $\pi_Y(\sigma_{X=\bar{a}}(D_R))$.

(a) For each $\bar{a} \in D_X$, build a K-D tree $T_{(X=\bar{a})}$ of $D_Y(X = \bar{a})$ by following step (a) in (1) above.

(b) For each ψ_i, its index on $D_Y(X = \bar{a})$ is a table $T^{\psi_i}_{(X=\bar{a})}(N, X, Y)$ consisting of 2^i tuples such that

(i) $\pi_N(T^{\psi_i}_{(X=\bar{a})}) = \{2^i\}$,

(ii) $\pi_X(T^{\psi_i}_{(X=\bar{a})}) = \{\bar{a}\}$, and

(iii) $\pi_Y(T^{\psi_i}_{(X=\bar{a})})$ consists of the 2^k tuples in the $(k+1)$-th level of K-D tree $T_{X=\bar{a}}$.

The resolution of index T^{ψ_i} for ψ_i on D_R can be calculated along the same lines as in case (1) above.

The same construction strategy applies to generic access templates of the form $R(X \to Y, N, \bar{d})$.

(3) Indices for access constraints. Following [11], for an access constraint $\psi = R(X \to Y, N, 0)$, its index on a relation instance D_R of R is simply a table $T^\psi(X, Y)$ built by first projecting D_R on attributes XY, yielding a table $T(X, Y)$, and then building a hash index on attribute X for T.

(4) A unified storage of indices for access schema. We next build indices for an access schema $\mathcal{A}$, by grouping indices for access templates and constraints in $\mathcal{A}$ together, to provide a unified index for query answering.

(a) We first extend each index table T_k^R of an access template ψ_k in $\mathcal{A}_t$ (built in case (1) above) with an extra column (attribute) X and let $\pi_X(T_k^R) = \{\varnothing\}$. Similarly, we also extend the index table T^ψ (built in case (3) above) of access constraint $\psi = R(X \to Y, N, 0)$ with a column (attribute) I such that $\pi_N(T^\psi) = \{N\}$.

(b) For access templates and constraints of the form $R(X \to Y, N, \bar{d})$, *i.e.,* sharing the same attributes X and Y, we take the union of all their index tables, yielding a unified index table $T_{R(XY)}(N, X, Y)$. Note that this is always possible since all these tables share the same relation schema (*i.e.,* attributes).

(c) We finally build a hash index on attributes NX for $T_{R(XY)}$, which provides index for fetching Y-values together with the associated resolution given any X-value and N. Note that NX uniquely identifies the index for an access template $R(X \to Y, N, \bar{d})$ in T.

K-D tree specification. We use a variant of K-D tree from [19], where all points are stored in leaves. We make the following adjustment to the K-D tree construction. (1) Each inner node carries a point randomly picked from its descendants. (2) We use the following split mechanism: let T be a K-D tree of tuples in D over R and let T_k be the set of nodes at the k-the level of T; for each attribute A in R and u in T_k, let $r_A(u) = [lb, ub]$, where lb (resp. ub) is the minimum (resp. maximum) A-value of all tuples carried by descendants of u; at level k, we choose attribute A of R such that $\mathsf{max}_{u \in T_k} r_A(u)$ is the maximum among all attributes of R; we then split each node u at the k-th level of T by binary partition with attribute A, using $\frac{lb+ub}{2}$.

One issue with this K-D tree is that the tree T may not be balanced, and thus T may not have 2^{k-1} nodes at the k-th level. To overcome this, after constructing the K-D tree T as above, we perform a traversal of T to lift nodes at $i+1$-th level with the smallest maximum value range to the i-th level if the i-the level is not full, *i.e.,* if it has fewer than 2^{i-1} nodes. Note that such variant of K-D tree can be constructed in $\tilde{O}(|D||R|)$-time, where $|D|$ is the number of tuples in D and $|R|$ is the number of attributes in R.

Justification. There are other possibly simpler and faster approaches to constructing the indices of $\mathcal{A}_t$, *e.g.,* random sampling. We adopt K-D trees for two reasons. (1) K-D trees are easy to implement and maintain [19]. (2) When zooming in to the next level, *i.e.,* when our budget allows us

to "upgrade" from access template ψ_k^R with 2^k representative tuples to ψ_{k+1}^R with 2^{k+1} tuples, K-D trees assure that we can maximize the gain in resolution $\bar{d}_{k+1} - \bar{d}_k$, as indicated by how $\bar{d}_k[B]$ is defined above. This allows us to fetch representative tuples from indices of an access schema and maximize the RC-measure of approximate answers, when the total number of tuple access is bounded under a small resource ratio α.

Indeed, the K-D tree variant specified above guarantees us an *near-optimal* refinement. Consider access templates $\psi_k = R(\varnothing \to \mathsf{attr}(R), 2^k, \bar{d}_k)$ and $\psi_{k+1} = R(\varnothing \to \mathsf{attr}(R), 2^{k+1}, \bar{d}_k)$ in $\mathcal{A}_t$. Let I_k be an index of ψ_k on a relation D_R of R with 2^k tuples and resolution $\bar{d}_k$. We say that an index I_{k+1} with 2^{k+1} tuples is a *zoom-in* of I_k if each tuple t in I_k is replaced by two tuples that are represented by t (*i.e.,* tuples in the leaf descendants of t in the K-D tree). By the K-D tree construction above and the definition of access templates, one can verify the following.

Proposition 2: *For any index I_{k+1} of ψ_{k+1} that is a zoom-in of index I_k of ψ_k on D of R, $2\mathsf{max}_{A\in\mathsf{attr}(R)}\bar{d}^*_{k+1}(A) \leqslant \mathsf{max}_{B\in\mathsf{attr}(R)}\bar{d}_{k+1}(B)$, where $\bar{d}^*_{k+1}$ is the resolution achieved by the K-D tree implementation of index for ψ_{k+1} above.* □

Intuitively, Proposition 2 ensures that the K-D tree approach above gives us a near-optimal index zooming-in strategy: the index for ψ_{k+1} obtained by zooming into I_k for ψ_k has resolution at most twice as large as the resolution of the optimal index that can be obtained by zooming into I_k.

Incremental maintenance. When D is updated with a sequence ΔD of tuple insertions, access schema $\mathcal{A}_t$ and its indices $\mathcal{I}_{\mathcal{A}_t}$ on D can be maintained by an incremental algorithm in $O(\|\mathcal{A}_t\||\Delta D|)$-time, where $\|\mathcal{A}_t\|$ is the number of templates in $\mathcal{A}_t$. The algorithm is *bounded* [36], *i.e.,* its cost is determined by the size of changes in the input and output, independent of possibly big dataset D. The result also holds on tuple deletions, which, together with insertion, can simulate value modifications. Below we give the details for tuple insertions. Tuple deletions and access templates beyond $\mathcal{A}_t$ can be handled along the same line.

Bounded maintenance. Consider access templates $\psi = R(\varnothing \to \mathsf{attr}(R), 2^k, \bar{d}_k)$ in $\mathcal{A}_t$, index $T^\psi(N, \mathsf{attr}(R))$ of ψ on a relation D_R of R (recall the implementation of index for ψ above), and a set ΔD of tuples. We show how to adjust ψ and T^ψ for $D \cup \Delta D$. For each $t' \in \Delta$, we do the following:

(1) randomly pick a tuple t in T^ψ;

(2) check the attribute differences between t and t'; if the differences are bounded by the resolution of T^ψ for ψ, then keep T^ψ and ψ unchanged; otherwise

(3) keep T^ψ, but change the resolution tuple $\bar{d}$ for ψ by replacing $\bar{d}[A]$ with $\mathsf{max}(\bar{d}[A], \Delta d[A])$, where $d[A]$ is the difference between t and t' on attribute A.

Observe that the process takes $O(|\Delta D|)$ time only since it keeps T^ψ unchanged and thus does not need to update the hash index on T^ψ. As a result, $\mathcal{A}_t$ and its indices on D can be updated in $O(\|\mathcal{A}_t\||\Delta D|)$ time, by processing each access templates in $\mathcal{A}_t$ one by one.

Calibration. Observe that the incremental algorithm above only updates the resolution tuple of ψ realized by T^ψ on $D \cup \Delta D$, while keeping the index T^ψ unchanged. While this assures us the bounded maintenance, it may degrades the optimality (recall Proposition 2) of T^ψ for $D \cup \Delta D$ if ΔD is big.

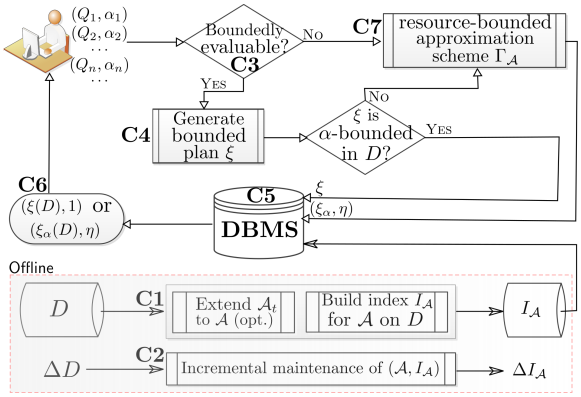


Figure 2: Complete workflow of BEAS

To handle this, we adopt a "batch-and-accumulate" approach. We record every processed updates $\Delta D_1, \ldots, \Delta D_m$ using the above bounded incremental algorithm. When $|\bigcup_{i=1}^m \Delta D_i|$ exceeds $\epsilon|D|$ for a predefined ϵ, we use database triggers to invoke a "calibration" process to adjust the index T^ψ for ψ, by (a) building an access template index ΔT^ψ for ψ on $\bigcup_{i=1}^m \Delta D_i$, and (b) merging T^ψ with ΔT^ψ.

Remarks. There are other approaches to maintaining access schema $\mathcal{A}_t$ and its indices on D in response to updates to D. For example, one can adopt dynamic algorithms for K-D trees maintenance [19], which can achieve better resolution with the price of higher maintenance cost.

4.2 BEAS: A Resource Bounded Framework

We next present BEAS, a framework for querying (big) relations. It is to be built on top of DBMS and extend DBMS with a functionality of resource-bounded query evaluation.

As shown in Fig. 2, BEAS consists of two parts. Given an application with dataset D, it works as follows.

(1) Offline algorithms. As offline preprocessing, algorithms for discovering access schema $\mathcal{A}$ can be plugged into BEAS, to enrich $\mathcal{A}_t$. BEAS builds indices $I_{\mathcal{A}}$ for $\mathcal{A}$ on D (**C1**). It also maintains $I_{\mathcal{A}}$ in response to updates to D (**C2**).

(2) Online algorithms. For any $\mathsf{RA}_{\mathsf{aggr}}$ query Q posed on D with a requested resource ratio α, BEAS first checks whether Q is boundedly evaluable under the set $\mathcal{A}_c$ of access constraints in $\mathcal{A}$ (**C3**). This is checked efficiently using an effective syntax for boundedly evaluable RA queries [11]. If so, it generates a boundedly evaluable query plan ξ (**C4**). It checks whether ξ is α-bounded in D. If so, it executes ξ directly using the underlying DBMS to compute exact answers $\xi(D)$ to Q in D, by accessing a bounded dataset $D_C \subseteq D$ with $|D_C| \leqslant \alpha|D|$ (**C5**), and returns $(\xi(D), 1)$ (**C6**). The algorithms for **C3**–**C5** have already been developed in [11].

If either Q is not bounded under $\mathcal{A}_C$ or the plan ξ in **C4** is not α-bounded in D, BEAS invokes a resource-bounded approximation scheme $\Gamma_{\mathcal{A}}$ to compute an α-bounded plan ξ_α for Q in D, and an accuracy bound η_α for ξ_α (Theorem 1; **C7**). It executes ξ_α by the DBMS, accessing at most $\alpha|D|$ tuples (**C5**). It returns $(\xi_\alpha(D), \eta)$ (**C6**). Underlying the approximation scheme are the algorithms for **C7**, which will be provided in Sections 5, 6 and 7. The α-bounded plans are executed directly by DBMS like exact bounded plans (**C5**).

BEAS is able to answer relational queries, *aggregate or not*, with limited resources. It makes *no assumption* on input queries, *i.e.,* it deals with *unpredictable* queries. Bounded

evaluation (**C3**–**C4**) is a bonus. While we invite interested reader to consult [11] about **C3**–**C4**, resource-bounded approximation (**C5**–**C7**) of this work alone is already able to efficiently answer queries with bounded resources.

Example 4: Recall $\mathcal{A}_0$, Q_1 and Q_2 of Example 1. Under schema $\mathcal{A}_1 = \mathcal{A}_t \cup \mathcal{A}_0$, BEAS answers Q_1 and Q_2 as follows.

(1) It finds that Q_1 is not boundedly evaluable under $\mathcal{A}_1$ (**C3**). Thus it generates an α-bounded plan ξ_α for Q_1 under $\mathcal{A}_1$, and an accuracy bound η (**C7**). The plan is carried out by DBMS, and returns approximate answers $\xi_\alpha(D)$ (**C6**).

(2) It finds that Q_2 is boundedly evaluable under $\mathcal{A}_1$ (**C3**), and thus generates a bounded query plan ξ for Q (**C4**). It checks whether ξ is α-bounded in D. If so, it executes ξ directly using DBMS (**C5**), and returns exact answers $\xi(D)$ with accuracy bound $\eta = 1$ (**C6**). Otherwise it answers Q_2 in the same way as Q_1 using the approximation scheme. □

5. Approximating SPC Queries

As a step to prove Theorem 1(2), we develop a resource-bounded approximation scheme for SPC queries (with σ, π, $\times$ and ρ operators). It works under any access schema $\mathcal{A}$ that subsumes $\mathcal{A}_t$ given in the proof of Theorem 1(a).

One might want to have an "optimal" approximation scheme that can find α-bounded plans with the maximum accuracy bound. This, however, is beyond reach in practice.

Theorem 3: *Given an* SPC *query Q, an access schema $\mathcal{A}$, a database D, a resource ratio α and a predefined bound η, it is Σ_3^p-hard to decide whether there exists an α-bounded plan ξ_α for Q in D under $\mathcal{A}$ with accuracy above η.* □

Here Σ_3^p is the complexity class at the third level of the polynomial hierarchy [39], beyond NP unless P = NP.

The hardness comes from the choices for what attributes to fetch, in what order to fetch them, and what access templates to use, to comply to resource ratio α and accuracy η.

Proof: We show that it is Σ_3^p-hard to decide whether there exists an α-bounded plan ξ_α for Q under $\mathcal{A}$ in D that achieves accuracy above η with the index for $\mathcal{A}$ in D, by reduction from the $\exists^*\forall^*\exists^*$CNF problem, which is known to be Σ_3^p-complete [39]. The $\exists^*\forall^*\exists^*$CNF problem is to decide, given a sentence $\phi = \exists X \forall Y \exists Z\, \psi(X, Y, Z)$, whether ϕ is true. Here $X = \{x_1, \ldots, x_m\}$, $Y = \{y_1, \ldots, y_n\}$, $Z = \{z_1, \ldots, z_o\}$ and ψ is a conjunction $C_1 \wedge \cdots \wedge C_r$ in which each C_i is a disjunction of three literals defined in terms of variables in $X \cup Y \cup Z$ or negations thereof. For convenience, we assume *w.l.o.g.* that $m = 2^k$ for some number k, *i.e.,* $\log_2 m$ is an integer, where m is the number of variables in X.

Given $\phi = \exists X \forall Y \exists Z\, \psi(X, Y, Z)$, we define an SPC query Q, an access schema $\mathcal{A}$, a database D, a resource ratio α and an accuracy bound η. We show that there exists an α-bounded plan ξ_α in D together with an index $I_\mathcal{A}$ on D for $\mathcal{A}$ such that $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D) \geqslant \eta$ and $\mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), Q, D) \geqslant \eta$ if and only if ϕ is true. We define D, $\mathcal{A}$, Q, α and η as follows.

(1) The database D consists of six relations specified by relation schemas $R_{01}(B)$, $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$, $R_\neg(A, \bar{A})$, $R_s(A)$ and $R_X(I, X)$. Their instances correspond to the first five relations shown in Fig. 5. More specifically, I_{01} encodes the Boolean domain, and $I_\vee$, $I_\wedge$ and $I_\neg$ encode disjunction, conjunction and negation, respectively, such that ψ can be expressed in SPC in terms of these relations. In addition, I_s will be used to differentiate between D and D_Q. Furthermore, I_X consists of $2m$ tuples of the form $(i, 0)$ and $(i, 1)$ for $i \in [1, m]$, to encode truth assignments of X. Note that $|D| = 2m + 14$. For each domain U over which attributes of D are defined, we define distance function $\mathsf{dis}_U(x, y) = 1$ if $x \neq y$ and $\mathsf{dis}_U(x, y) = 0$ otherwise.

(2) The access schema $\mathcal{A}$ consists of only access templates in $\mathcal{A}_t$ specified in the proof of Theorem 1; we use no additional access constraints in the proof.

(3) We define the SPC query Q as follows, expressed in the syntax of relational calculus:

$$Q(\bar{y}, \bar{u}) = \exists \bar{x}, \bar{z}, w \big(Q_X(\bar{x}) \wedge Q_Y(\bar{y}) \wedge Q_Z(\bar{z}) \wedge Q_\psi(\bar{x}, \bar{y}, \bar{z}, w) \wedge R_s(w) \wedge Q_{\mathsf{all}}(\bar{u}) \wedge R_s(1)\big).$$

Here Q_Y and Q_Z generate all truth assignments of Y and Z variables, respectively, by means of Cartesian products of R_{01}. Furthermore, $Q_X(\bar{x}) = \bigwedge_{i=1}^m R_X(i, x_i)$ selects truth assignments of X from I_X, and we use $Q_{\mathsf{all}}(\bar{u}) = R_{01}(u_1) \wedge R_\wedge(u_2, u_3, u_4) \wedge R_\vee(u_5, u_6, u_7) \wedge R_\neg(u_8, u_9)$, to ensure that all tuples in I_{01}, $I_\wedge$, $I_\vee$ and $I_\neg$ have to be carried over to D_Q. Similarly, $R_s(1)$ is included to avoid the deletion of (1) from I_s in D_Q. Finally, sub-query Q_ψ in SPC encodes the truth value of $\psi(X, Y, Z)$ for given truth assignments μ_X, μ_X and μ_Z, in terms of $I_\vee$, $I_\wedge$ and $I_\neg$. More specifically, $Q_\psi(\mu_X, \mu_Y, \mu_Z, w)$ returns $w = 1$ if $\psi(X, Y, Z)$ is satisfied by μ_X, μ_Y and μ_Z, and $w = 0$ otherwise.

Intuitively, query Q returns all truth assignments μ_Y of Y for which there exists a truth assignment μ_X of X and a truth assignment μ_Z of Z such that $Q_\psi(\mu_X, \mu_Y, \mu_Y, w) \wedge R_s(w)$ holds and in addition, Q pairs such truth assignments with all tuples in I_{01}, $I_\wedge$, $I_\vee$ and $I_\neg$.

(4) We set $\alpha = \frac{13+m}{14+2m}$. Recall that m is number of variables in X. The number 13 stems from the sum of the sizes of the instances I_{01}, $I_\wedge$, $I_\vee$ and $I_\neg$ and the requirement that we consider only instances of R_s that contain (1).

(5) We let $\eta = 1$.

We verify that ϕ is true if and only if that there exist an α-bounded plan ξ_α under $\mathcal{A}$ for Q in D and an index $I_\mathcal{A}$ for $\mathcal{A}$ on D such that $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D) = \mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), Q, D) = 1 = \eta$ with $I_\mathcal{A}$. That is, the construction above is a reduction.

$\boxed{\Rightarrow}$ Observe that $Q(D)$ returns $Q_Y(D) \times Q_{\mathsf{all}}(D)$ since $I_s = \{(0), (1)\}$, and thus truth assignments of Y are returned independent of the validity of ϕ. Observe that $Q(D) \neq \varnothing$. Thus, by the definition of RC-measure, if $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D) = \mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), Q, D) = \eta = 1$, then $\xi_\alpha(D) = Q(D)$. Let D_{ξ_α} be the set of accessed tuples by ξ_α in D. Then $\xi_\alpha(D) = Q(D_{\xi_\alpha})$ since $\eta = 1$. By the definition of Q, whenever D_{ξ_α} is obtained from D by removing tuples from I_{01}, $I_\wedge$, $I_\vee$ and $I_\neg$, then $\xi_\alpha(D) = Q(D_{\xi_\alpha}) \neq Q(D)$. Similarly, when D_{ξ_α} does not contain (1) in I'_s, then $\xi_\alpha(D) = Q(D_{\xi_\alpha}) = \varnothing$. Hence, if ξ_α accesses only D_{ξ_α} of D and in addition, $Q(D) = \xi_\alpha(D) = Q(D_{\xi_\alpha})$, then ξ_α can only miss tuples from I_X and tuple (0) from I_s, and has to fetch every tuple in other relations of D. In addition, note that $\xi_\alpha(D) = Q(D_{\xi_\alpha}) = \varnothing$ if there exists $i \in [1, m]$ such that neither $(i, 0)$ nor $(i, 1)$ is present in D_{ξ_α}. Thus, D_{ξ_α} must be of cardinality at least $13 + m$. Since $\alpha = \dfrac{13 + m}{14 + 2m}$ and $|D| = 14 + 2m$, this implies that D_{ξ_α} agrees with D on I_{01}, $I_\wedge$, $I_\vee$ and $I_\neg$, $I'_s = \{(1)\}$, and I'_X consists of m tuples (i, x_i) for $i \in [1, m]$. Here we use I'_s to denote the instance of R_s in D_{ξ_α}; similarly for I'_X.

I_{01} =

X
1
0

$I_\vee$ =

B	A_1	A_2
0	0	0
1	0	1
1	1	0
1	1	1

$I_\wedge$ =

B	A_1	A_2
0	0	0
0	0	1
0	1	0
1	1	1

$I_\neg$ =

A	$\bar{A}$
0	1
1	0

I_s =

A
0
1

Figure 3: Relation instances used in the proof of Theorem 3.

We claim that if $Q(D) = \xi_\alpha(D) = Q(D_{\xi_\alpha})$ then the truth assignment μ_X of X encoded in I'_X witnesses that $\forall Y \exists Z \psi(\mu_X, Y, Z)$ is true. Indeed, $Q(D) = Q(D_{\xi_\alpha})$ implies that all truth assignments of Y are returned by $Q(D_{\xi_\alpha})$, and furthermore, that for each such truth assignment μ_Y of Y, there exists a truth assignment μ_Z of Z such that $Q_\psi(\mu_X, \mu_Y, \mu_Z, w) \wedge R_s(w)$ holds. Since $I'_s = \{(1)\}$, this means that $\psi(\mu_X, \mu_Y, \mu_Z)$ must evaluate to true.

$\boxed{\Leftarrow}$ Conversely, assume that φ is true. Let μ_X^0 be a truth assignment of X that witnesses that $\forall Y \exists Z \psi(\mu_X, Y, Z)$ is true. Then we construct index $I_\mathcal{A}$ on D for $\mathcal{A}$ as follows.

- The index for access template $\psi_S = S(\varnothing \to \mathsf{attr}(S), 2^k, \bar{d}_{\mathsf{attr}(S)})$ for relations I_{01}, $I_\vee$, $I_\wedge$, $I_\neg$ are constructed by arbitrarily sampling tuples for each k. For instance, when S is $R_\vee$, $\mathsf{fetch}(\varnothing, R_\vee, BA_1A_2, 2^0, \psi_{R_\vee})$ can simply return (0, 0, 0).
- The index for template $\psi_{R_s} = R_s(\varnothing \to A, 2^k, \bar{d}_A)$ is constructed as follows: when $k = 0$, it returns 1; and when $k = 1$, it retrieves $\{0, 1\}$.
- The index for $\psi_{R_X} = R_X(\varnothing \to IX, 2^k, \bar{d}_{IX})$ is as follows: for $k = \log_2^m$, it returns $(i, \mu_X^0(x_i))$ for $i \in [1, m]$; and for any other k, it returns arbitrary 2^k tuples.

With such index $I_\mathcal{A}$ for $\mathcal{A}$ on D, an α-bounded plan ξ_α in D for Q works by first sampling relations in Q via fetch operations under $\mathcal{A}$ with $I_\mathcal{A}$, and then executing Q directly on the samples, where

- for fetch operations on relations other than R_s and R_X, set k to be their maximum;
- for $\mathsf{fetch}(\varnothing \to A, R_s, 2^k, \psi_{R_s})$, set $k = 0$; and
- for $\mathsf{fetch}(\varnothing \to IX, R_X, 2^k, \psi_{R_X})$, set $k = \log_2 m$.

Then ξ_α is an α-bounded plan for Q in D as it accesses $\alpha|D| = 13 + m$ tuples. Moreover, $\xi_\alpha(D) = Q(D_{\xi_\alpha}) = Q(D)$ as μ_X^0 makes φ true, *i.e.,* all truth assignments to Y are returned. Therefore, $\mathsf{F_{rel}}(\xi_\alpha(D), Q, D) = \mathsf{F_{cov}}(\xi_\alpha(D), Q, D) = \eta = 1$. □

Despite the challenge, below we provide a PTIME approximation scheme for SPC with a deterministic bound η.

Approximation scheme $\mathsf{BEAS_{SPC}}$. The scheme is denoted by $\mathsf{BEAS_{SPC}}$ and shown in Fig. 4. Given an SPC query Q, an access schema $\mathcal{A}$ that subsumes $\mathcal{A}_t$, and budget $B = \alpha|D|$, $\mathsf{BEAS_{SPC}}$ computes an α-bounded plan ξ_α for Q in D and a ratio η such that $\mathsf{accuracy}(\xi_\alpha(D), Q, D) \geqslant \eta$. Here $\alpha \in (0, 1]$ is a resource ratio, and $|D|$ is the size of database D to be queried. Note that D itself is *not* part of the input.

$\mathsf{BEAS_{SPC}}$ generates plan ξ_α in a *canonical* form (ξ_F, ξ_E), where (a) ξ_F is a *fetching plan* for Q under $\mathcal{A}$, which is a sequence of fetch operations, and (b) ξ_E is an *evaluation plan* for Q, which performs (relaxed) relational operations of Q. That is, ξ_α first fetches necessary data D_Q from D by accessing at most $\alpha|D|$ amount of data, and then ξ_E computes (approximate) answers $Q(D_Q)$ using D_Q. Every SPC query has a bounded query plan in this "normal form".

Algorithm $\mathsf{BEAS_{SPC}}$

Input: SPC query Q, access schema $\mathcal{A} \supseteq \mathcal{A}_t$, and $B = \alpha|D|$.
Output: An α-bounded plan ξ_α for Q under $\mathcal{A}$ and bound η.

1. $\ell := \mathsf{chase}(Q, \mathcal{A}, B)$; /* ℓ *is a chasing sequence for* Q */
2. generate a fetching plan ξ_F for Q from ℓ;
3. generate an evaluation plan ξ_E for Q *w.r.t.* ξ_F;
4. $(\xi_F^\alpha, \eta) = \mathsf{chAT}(\xi_F, Q, B, \mathcal{A})$;
5. **return** $(\xi_\alpha = (\xi_F^\alpha, \xi_E), \eta)$;

Procedure chAT

Input: ξ_F, Q, B and access schema $\mathcal{A}$;
Output: An α-bounded fetching plan ξ_F^α and accuracy bound η.

1. **while** $\mathsf{tariff}(\xi_F) \leqslant \alpha|D|$ **do**
2. $\quad \xi_\alpha := (\xi_F, \xi_E)$; $\eta := L(\xi_\alpha)$;
3. $\quad$ find δ such that $L_{\psi_k^\delta \to \psi_{k+1}^\delta}(\xi_\alpha) \geqslant L_{\psi_k^{\delta'} \to \psi_{k+1}^{\delta'}}(\xi_\alpha)$ for any δ';
4. $\quad$ replace template ψ_k^δ with ψ_{k+1}^δ for fetch δ in ξ_α;
5. **return** (ξ_α, η);

Figure 4: Algorithm $\mathsf{BEAS_{SPC}}$

Lemma 4: *Under any access schema $\mathcal{A} \supseteq \mathcal{A}_t$, every* SPC *query Q has a canonical bounded plan.* □

Proof: Under any access schema $\mathcal{A} \supseteq \mathcal{A}_t$, every SPC query Q has a canonical bounded plan $\xi = (\xi_F, \xi_E)$ as follows: (a) for each relation R that occurs in Q, ξ_F includes a fetch operation $\mathsf{fetch}(\varnothing, R, \mathsf{attr}(R), N, \bar{d})$ in $\mathcal{A}_t$ that fetches a sample of relation R; and (b) the evaluation plan ξ_E is simply the relational algebra expression of Q. Obviously, when all access templates used in ξ_F are upgraded to those ones with $\bar{d} = \bar{0}$, ξ is simply the algebra expression of Q. Therefore, ξ is a bounded plan for Q under $\mathcal{A}$. □

$\mathsf{BEAS_{SPC}}$ works in two steps. (1) It generates an initial α-bounded plan $\xi_\alpha = (\xi_F, \xi_E)$, where ξ_F uses either access constraints or templates with $k = 0$ as placeholders. (2) It then improves the accuracy of ξ_α by picking the best templates ψ_k (with largest N) for ξ_F, and keeps it α-bounded in D.

Step (1): getting initial α-bounded plan $\xi_\alpha = (\xi_F, \xi_E)$. To compute ξ_F, $\mathsf{BEAS_{SPC}}$ builds a *chasing sequence* ℓ for Q under $\mathcal{A}$ via $\mathsf{chase}(Q, \mathcal{A}, B)$ (line 1, Fig. 4), from which ξ_F is derived, observing $B = \alpha|D|$ (line 2). We revise the *chase*, a classical tool for query optimization with dependencies [6], such that each step in ℓ corresponds to a fetch operation in ξ_F that uses an access constraint or a template in $\mathcal{A}$.

The chase is defined on the tableau of Q. The *tableau* of an SPC Q is a pair $(T(Q), u(Q))$, where (a) $T(Q)$ is a collection of tables in which tuple templates represent relation atoms in Q; and (b) $u(Q)$ is a tuple of variables specifying the output of Q. That is, $T(Q)$ is a set of *tuple templates* with variables to be mapped to attribute values, and $u(Q)$ denotes projected attributes. Computing $Q(D)$ is essentially to fetch tuples in D, match tuple templates in $T(Q)$ and instantiate output tuple $u(Q)$ (see [6] for details of the chase).

For example, an SPC query Q_3 and its tableau are at the top of Fig. 5, in which constants and variables represent

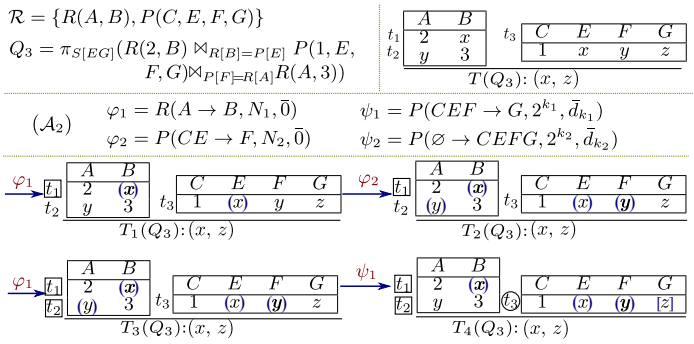


Figure 5: A chasing sequence for Q_3 under $\mathcal{A}_2$

attributes and specify the selection conditions in Q, *e.g.*, x encodes $R[B]$ and $P[E]$, and y denotes $P[F]$ and $R[A]$.

Chase (line 1). More specifically, a *chasing sequence* for Q under $\mathcal{A}$ is a sequence of annotated tableaux of $T(Q)$ of Q:

$$T_0(Q) \xmapsto{\gamma_0} T_1(Q) \xmapsto{\gamma_1} \cdots \xmapsto{\gamma_{m-1}} T_m(Q),$$

where each step indicates a fetch operation that retrieves attribute values to instantiate variables in $T(Q)$, using an access constraint or template in $\mathcal{A}$. More specifically, (a) $T_i(Q)$ is the same as $T(Q)$, including all the tuples of $T(Q)$, except that it marks some tuples and variables *exactly* or *approximately* covered (enclosed in square or circle for tuples, and parentheses or square brackets for variables in Fig. 5); and (b) γ_i is an access constraint or a template with $k = 0$ in $\mathcal{A}$ that is applied to $T_i(Q)$ and triggers the marking.

The chasing starts with $T_0(Q) = T(Q)$, without any mark. Each step $T_i(Q) \xmapsto{\gamma_i} T_{i+1}(Q)$ applies $\gamma_i = R(X \to Y, N, \bar{d}_i)$ to $T_i(Q)$ by one of the following two rules, which either marks a tuple template t in $T_i(Q)$ or a variable of t, to get $T_{i+1}(Q)$.

(a) Variable marking. If $t[X]$ (possibly empty) consists of constants or variables marked in prior steps, then variables in $t[Y]$ that are not yet exactly covered are marked (i) *exactly covered* if no variables in $t[X]$ are approximately covered and γ_i is a constraint, and (ii) *approximately covered* otherwise.

More specifically, if (a) $t[X]$ consists of constants or exactly covered variables, there are variables in $t[Y]$ that are not exactly covered, and if γ_i is an access constraint, then mark all variables in $t[Y]$ exactly covered. Otherwise if $t[X]$ consists of constants or covered variables (exactly or approximately), and if there are variables in $t[Y]$ that are neither exactly covered or approximately covered, then mark all variables in $t[Y]$ that are not yet covered (exactly or approximately) as approximately covered variables. Intuitively, this ensures that an approximately covered variable can possibly be "upgraded" to exactly covered but not the other way around.

(b) Tuple marking. If all *nontrivial variables* of $(T(Q), u(Q))$ in t are exactly covered by γ_i and if t is not yet exactly covered, then mark tuple template t *exactly covered*; otherwise if all its nontrivial variables are covered (approximately or exactly) and if t is not yet covered (exactly or approximately), then mark t *approximately covered*.

Here a variable x in $(T(Q), u(Q))$ is nontrivial if it occurs in either $u(Q)$ or multiples times in $T(Q)$. For example, all variables in Q_3 of Figure 5 are nontrivial variables Intuitively, one only needs to know the values of nontrivial variables to answer Q, regardless of the values of trivial variables.

Not that an approximately covered tuple can also be "upgrade" to exactly covered, but not the other way around.

Intuitively, exactly covered variables can be instantiated by a sequence of fetch operations with access constraints only, while those approximately covered involve fetch with access templates. A tuple is covered if its value can be retrieved via fetch, exact or approximate depending on its mark.

One can verify the following.

Lemma 5: *Under any $\mathcal{A} \supseteq \mathcal{A}_t$, for any* SPC *query Q, all chasing sequences terminate with the same $T_m(Q)$, in which all tuple templates are covered, exactly or approximately.* □

Proof: We first show that any chasing sequence always terminates in $2|Q| + 2\|Q\|$ steps, where $|Q|$ and $\|Q\|$ are the size and cardinality (the number of rows) of $T(Q)$, respectively. This follows from the following observations. (a) Every step of a chasing sequence marks a variable or tuple in $T(Q)$. (b) There are at most $|Q|$ variables and $\|Q\|$ tuples. (c) Each variable and tuple can be marked at most twice.

We next prove that the chasing always terminates with the same $T_m(Q)$, with all tuple templates covered.

(a) We first show that for any two chasing sequences ℓ and ℓ' for Q with $\mathcal{A}$ (any access schema), the set of exactly covered variables in $T(Q)$ is the same. Suppose by contradiction that ℓ and ℓ' lead to different sets of exactly covered variables in $T(Q)$, say S and S', respectively, and $S \neq S'$. Assume *w.l.o.g.* that $S \not\subseteq S'$. Then there must exist a variable x in $S \setminus S'$ and a step l_i in ℓ with constraint $\gamma_i = R(X \to Y, N, \bar{0})$ (X is possibly empty) that deduces x from constants and variables that are in $S \cap S'$, *i.e.,* there exists $t \in T_i(Q)$ of R with $t[X]$ consisting of constants or exactly covered variables and $x \in t[Y]$. Indeed, if this is not the case, then no variables in $S \setminus S'$ can be exactly covered by applying chasing rule (1). Since $x \notin S'$, we can append l_i with γ_i at the end of ℓ' to mark x as exactly covered. This contradicts the assumption that ℓ' is a chasing sequence. Thus $S = S'$.

Similarly, one can verify that ℓ and ℓ' end up with the same set of approximately covered variables. By $\mathcal{A} \supseteq \mathcal{A}_t$, all the variables that are not exactly covered must be approximately covered in $T_m(Q)$. Thus all chasing sequences terminate with $T_m(Q)$ that share the same variable markings.

(b) We next show that the sets of exactly and approximately covered tuple templates in $T(Q)$ by ℓ and ℓ' are the same. This is obvious given that the set of constants and (approximately and exactly) covered variables by ℓ and ℓ' are the same, as guaranteed by (a) and the chasing rule. Hence all chasing sequences terminate with the same $T_m(Q)$, in which all tuple templates are exactly or approximately covered. □

Chasing orders. While there are exponentially many chasing sequences in Lemma 5 ensures that all chasing sequences converge at the same annotated $T_m(Q)$ within $2\|Q\| + 2|Q|$ steps. Therefore, the order of applying chasing rules does not make a big difference in terms of the complexity of computing $T_m(Q)$. In each chasing step, we simply pick the rule that exactly covers the maximum number of variables when multiple applications of the marking rule are available.

Example 5: Consider query Q_3 and access schema $\mathcal{A}_2$ given in Fig. 5. A chasing sequence of 5 steps for Q_3 under $\mathcal{A}_2$ is depicted at the bottom of Fig. 5. It marks the following: (1) x is exactly covered (by access constraint φ_1); (2) t_1 is exactly covered (by φ_1 since variable x is covered); (3) y is exactly covered (by access constraint φ_2 with exactly covered variable x); (4) t_2 is exactly covered (by φ_1 with exactly

covered variable y); (5) z is approximately covered (by access template ψ_1); and (6) t_3 is approximately covered (by ψ_1). In Fig. 5, chase steps (1) and (2) are shown together in a single step (the first step); similarly for (5) and (6). □

The process also maintains the total number tariff of tuples accessed. It is estimated by means of constants N in access constraints applied, *without* accessing D. Applying a constraint increases tariff with the number of tuples fetched; if tariff exceeds budget B, we use template $R(\varnothing \to \mathsf{attr}(R), 2^0, \bar{d}_0)$ instead. Each template ψ_k applied increases the count tariff only by 1 since we start with $k = 0$.

Fetching plan from chase (line 2). Given a chasing sequence ℓ, an α-bounded fetching plan for Q in D is derived as follows. (1) For each tuple t_R of relation R in $T(Q)$, let $T_i(Q) \xmapsto{\gamma_i} T_{i+1}(Q)$ be the chase step that marks t_R covered. Then a fetching plan ξ_F^R for relation R in Q includes $(T_1 = \xi(V), T_2 = \mathsf{fetch}(X \to T_1, R, Y, \gamma_i))$, where V is the set of constants and covered variables for $T_1[X]$, and $\xi(V)$ is the plan that fetches variables in V (possibly using $\times$ and π_Y). (2) The fetching plan ξ_F for Q under $\mathcal{A}$ simply collects all such ξ_R^F for all relation names R used in Q.

Intuitively, for each relation R in Q, the part of chasing sequence that marks t_R of R in $T(Q)$ covered is directly translated into a fetching plan for R, as illustrated below.

Example 6: From the chasing sequence of Fig. 5, a fetching plan $\xi_F^{Q_3}$ for Q_3 under $\mathcal{A}_2$ is derived as follows:

$T_1 = \mathsf{fetch}(A \in \{2\}, R, B, \varphi_1)$ (/*fetch x and t_1 of R*/);
$T_2 = \mathsf{fetch}(CE \in \{1\} \times \pi_B(T_1), P, F, \varphi_2)$ (/*fetch y*/);
$T_3 = \mathsf{fetch}(A \in \pi_F(T_2), R, B, \varphi_1)$ (/*fetch t_2 of R*/);
$T_4 = \mathsf{fetch}(CEF \in T_2, P, G, \psi_1)$ (/*fetch z and t_3 of P*/);

where $\xi_F^{R(2,B)} = T_1$, $\xi_F^P = (T_1, T_2, T_4)$, $\xi_F^{R(A,3)} = (T_1, T_2, T_3)$. Here T_1 fetches both variable y and R tuple t_1 in $T(Q_3)$. □

Evaluation plan ξ_E (line 3). The evaluation plan computes answers to relaxed query Q_r of Q using D_Q fetched by ξ_F. It performs the same relational operations in Q except that it (1) uses fetched tuples for each relation in Q; and (2) relaxes selection conditions on attributes fetched via access templates ψ to accommodate the resolution of ψ:

(a) For each selection $\sigma_{A=c}$, if A is fetched via an access template $R(X \to Y, N, \bar{d})$ in $\mathcal{A}$, where $A \in Y$ and c is a constant, then replace it with $\sigma_{|\mathsf{dis}_A(A,c)| \leqslant \bar{d}[A]}$.
(b) For each $\sigma_{A=B}$, assume that A and B are fetched via access templates with resolution tuples $\bar{d}_1$ and $\bar{d}_2$, respectively, then replace it with $\sigma_{|\mathsf{dis}(A,B)| \leqslant \bar{d}_1[A] + \bar{d}_2[B]}$.

Note that only attributes that are fetched via access templates and used in selections are compensated by relaxation with resolution, as opposed conventional approximate joins. The targeted relaxation guarantees an accuracy bound.

For instance, given the fetching plan of Example 6, the evaluation plan for Q_3 under $\mathcal{A}_2$ conducts the same operations in Q_3. No relaxation is needed since all attributes in its selections are fetched with access constraints only. While t_3 is approximately covered, it does not affect the selections.

Step (2): choosing access templates for ξ_F. BEAS$_{\mathsf{SPC}}$ next optimizes ξ_α by choosing access templates ψ_k with higher resolution (*i.e.,* larger N, smaller $\bar{d}$) to increase its accuracy, by invoking procedure chAT (line 4 of Fig. 4).

Given $\xi_\alpha = (\xi_F, \xi_E)$, budget $B = \alpha|D|$ and access schema $\mathcal{A}$, chAT iteratively identifies a fetch operation δ in ξ_F such that using access templates with higher resolution yields the maximum accuracy improvement among all fetch operations in ξ_F (lines 1-4). The tricky part is to assess the accuracy with ξ_α and $\mathcal{A}$ only, without accessing D. To achieve this, chAT uses a function L (to be given below) that computes a lower bound for the accuracy of ξ_α in D.

More specifically, in each iteration, chAT first sets $\eta = L(\xi_F)$ (line 2), and then identifies a fetch operation δ in ξ_F with access template $\psi_k^\delta = R(X \to Y, 2^k, \bar{d}_k)$ such that replacing ψ_k^δ with $\psi_{k+1}^\delta = R(X \to Y, 2^{k+1}, \bar{d}_{k+1})$ yields the maximum accuracy improvement (break ties arbitrarily; line 3), *i.e.,* $L_{\psi_k^\delta \to \psi_{k+1}^\delta}(\xi_F) \geqslant L_{\psi_k^{\delta'} \to \psi_{k+1}^{\delta'}}(\xi_F)$ for any other fetch δ' in ξ_F, where $L_{\psi_k^\delta \to \psi_{k+1}^\delta}(\xi_F)$ denotes $L(\xi_F')$, and ξ_F' revises ξ_F by replacing ψ_k^δ with ψ_{k+1}^δ. It replace ψ_k^δ in ξ_α with ψ_{k+1}^δ (line 4). This proceeds until the estimated amount of data accessed by ξ_α (denoted by $\mathsf{tariff}(\xi_\alpha)$) exceeds the budget B (line 1) (see Appendix A for the detailed definition of $\mathsf{tariff}(\xi_\alpha)$). It returns (ξ_α, η) after the iteration (line 5).

Lower bound function $L(\xi_F)$. Function L is given as $1/(1 + \mathsf{max}(d_{\mathsf{rel}}, d_{\mathsf{cov}}))$, where d_{rel} and d_{cov} are upper bounds on the relevance and coverage distances of approximate answers.

The upper bounds are inductively defined based on the structure of query Q *w.r.t.* the fetching plan ξ_F of ξ_α.

(1) Q is relation R. If R is fetched in ξ_α by a fetch operation with access template $R(\varnothing \to \mathsf{attr}(R), N, \bar{d})$, then

- $d_{\mathsf{rel}}(Q) = 0$.
- $d_{\mathsf{cov}}(Q) = \sum_{A \in \mathsf{attr}(R)} \bar{d}[A]$.

That is, every approximate answer to R has relevance measured by $d_{\mathsf{rel}}(Q)$, and their coverage distance is no more than the sum of the "errors" on all the attributes of R, which are bounded by resolution $\bar{d}$ in the access template used.

(2) Q is $\pi_Y(Q')$. We have that

- $d_{\mathsf{rel}}(Q) = d_{\mathsf{rel}}(Q')$.
- $d_{\mathsf{cov}}(Q) = \sum_{A \in Y} d_{\mathsf{cov}}(Q')[A]$.

That is, the largest relevance distance of any approximate answer s to Q inherits from the relevance of s to Q'. The coverage distance bound is the sum of the coverage distance for Q' *w.r.t.* attributes Y, since the result schema R_Q of query Q consists of Y-attributes only.

(3) Q is $Q_1 \times Q_2$.

- $d_{\mathsf{rel}}(Q) = d_{\mathsf{rel}}(Q_1) + d_{\mathsf{rel}}(Q_2)$.
- $d_{\mathsf{cov}}(Q) = d_{\mathsf{cov}}(Q_1) + d_{\mathsf{cov}}(Q_2)$.

Intuitively, the worst-case relevance distance is the sum of distances for Q_1 and for Q_2; similarly for coverage distance.

(4) Q is $\sigma_{R[A]=R'[B]}(Q')$.

- $d_{\mathsf{rel}}(Q) = d_{\mathsf{rel}}(Q') + \bar{d}_\delta[A] + \bar{d}_{\delta'}[B]$.
- $d_{\mathsf{cov}}(Q) = d_{\mathsf{cov}}(Q')$.

Here δ and δ' are the fetch operations that retrieve R and R' data, respectively, and $\bar{d}_\delta$ and $\bar{d}_{\delta'}$ are the resolution tuples in the access templates used by the fetch operations, respectively. Note that when $R[A]$ (resp. $R[B]$) is fetched by an access constraint, $\bar{d}_\delta[A] = 0$ (resp. $\bar{d}_{\delta'}[B] = 0$).

When Q is $\sigma_{R[A] \leqslant c}(Q')$ or $\sigma_{R[A] \leqslant R'[B]}(Q')$, $d_{\mathsf{rel}}(Q)$ and $d_{\mathsf{cov}}(Q)$ are the same as the cases when Q is $\sigma_{R[A]=c}(Q')$ or $\sigma_{R[A]=R'[B]}(Q')$, respectively.

Example 7: Given Q_1 and $\mathcal{A}_0$ from Example 1, BEAS$_{\mathsf{SPC}}$

returns exactly the plan in Example 1 for Q_1 under $\mathcal{A}_0 \cup \mathcal{A}_t$, and $\eta = 1/(1 + e_p^{k_\alpha} + e_a^{k_\alpha})$. After fetching 10000 friend and person tuples via constraints, it probes templates on poi one by one starting from ψ_1, until it gets to ψ_{k_α}, which reaches the budget $\alpha|D_0|$. It picks ψ_{k_α} and adjusts d_{cov} (resp. d_{rel}) from 0 to $e_p^{k_\alpha} + e_a^{k_\alpha}$ (resp. $e_p^{k_\alpha}$), which gives the η. □

Analysis. We show the following about $\mathsf{BEAS}_{\mathsf{SPC}}$.

Theorem 6: *For any* SPC *query Q, access schema $\mathcal{A} \supseteq \mathcal{A}_t$, resource ratio α and dataset $D \models \mathcal{A}$, $\mathsf{BEAS}_{\mathsf{SPC}}$ generates an α-bounded plan ξ_α and a bound η for Q under $\mathcal{A}$ in $O(|Q|\,\mathsf{min}(\|\mathcal{A}\|, \|Q\| \log \alpha|D|))$-time, without accessing D, such that*

(1) $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D) \geqslant \eta$ and $\mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), Q, D) \geqslant \eta$;

(2) $\eta \geqslant 1/(1 + \mathsf{max}(d^*_{\mathsf{rel}}, d^*_{\mathsf{cov}}))$, where
- *$d^*_{\mathsf{rel}} = (\mu_C(Q) + 2\mu_v(Q)) \cdot \mathsf{max}_{\varphi \in \mathcal{A}_0} \bar{d}^m_{(\psi,k^*)}$,*
- *$d^*_{\mathsf{cov}} = \nu(Q) \cdot \mathsf{max}_{\varphi \in \mathcal{A}} \bar{d}^m_{(\psi,k^*)}$,*

in which $k^* = \lfloor \log_2 \frac{\alpha|D|}{\|Q\|} \rfloor - 1$; and

(3) if $\alpha_1 \geqslant \alpha_2$, then $\eta_1 \geqslant \eta_2$, where η_1 and η_2 are the accuracy bounds returned by $\mathsf{BEAS}_{\mathsf{SPC}}$ for Q in D with resource ratios α_1 and α_2, respectively. □

Here (1) for each access template $\psi_k = R(X \to Y, 2^k, \bar{l}_k)$ in $\mathcal{A}_0$, we denote $\mathsf{max}_{A \in Y} \bar{l}_k[A]$ as $\bar{d}^m_{(\psi,k)}$; (2) for a query Q, $\|Q\|$ is the number of relations in Q, and $|Q|$ is the size of Q; $\mu_C(Q)$ (resp. $\mu_v(Q)$) is the number of constant (resp. equality) selections in Q; and $\nu(Q)$ is the arity of the output schema R_Q of Q; (3) $\|\mathcal{A}\|$ is the cardinality of $\mathcal{A}$.

Observe the following. (1) The bound η returned by $\mathsf{BEAS}_{\mathsf{SPC}}$ is always correct and above 0, as it is above $1/(1 + \mathsf{max}(d^*_{\mathsf{rel}}, d^*_{\mathsf{cov}})) > 0$. This verifies Theorem 1 for SPC. (2) The lower bound $1/(1 + \mathsf{max}(d^*_{\mathsf{rel}}, d^*_{\mathsf{cov}}))$ for η depends on the structure of Q, since $\mathsf{BEAS}_{\mathsf{SPC}}$ advocates dynamic data reduction instead of an "one-size-fit-all" synopsis (see Fig. 1). (3) As will be seen in Section 8, $\mathsf{BEAS}_{\mathsf{SPC}}$ returns η much higher than $1/(1 + \mathsf{max}(d^*_{\mathsf{rel}}, d^*_{\mathsf{cov}}))$. This is because $\mathsf{BEAS}_{\mathsf{SPC}}$ computes η based on the real resolutions of the access templates actually used in ξ_α, instead of d^*_{rel} and d^*_{cov} estimated above by using Q alone. (4) $\mathsf{BEAS}_{\mathsf{SPC}}$ may not use all templates in $\mathcal{A}$, as evidenced by $\mathsf{min}(\|\mathcal{A}\|, \log \alpha|D|)$ in its complexity.

We next prove Theorem 6.

Proof: We prove Theorem 6 in three steps. We first show that $\mathsf{BEAS}_{\mathsf{SPC}}$ correctly returns an α-bounded plan for Q under $\mathcal{A}$. We then verify its time complexity. Finally we prove its performance guarantees as stated in Theorem 6.

(A) Correctness of $\mathsf{BEAS}_{\mathsf{SPC}}$. To show that $\mathsf{BEAS}_{\mathsf{SPC}}$ is correct, we prove the following: *for any* SPC *query Q and any access schema $\mathcal{A}$, if there exists a chasing sequence ℓ that covers every tuple in $T(Q)$, then the canonical plan (ξ_F, ξ_E)* w.r.t. *ℓ is a bounded query plan for Q under $\mathcal{A}$.*

By Lemma 5, we know that line 1 of $\mathsf{BEAS}_{\mathsf{SPC}}$ (Fig. 4) is always feasible. Furthermore, by the property above, we know that lines 2-3 of $\mathsf{BEAS}_{\mathsf{SPC}}$ are feasible and the initial plan (ξ_F, ξ_E) is an α-bounded plan for Q under $\mathcal{A}$. Since line 4 improves the accuracy of (ξ_F, ξ_E) while keeping it α-bounded in D, and since it does not change the order of attributes fetched by ξ_F, the final plan ξ_α returned is an α-bounded plan for Q under $\mathcal{A}$. We prove the property below.

We show that (ξ_F, ξ_E) is a query plan for Q under $\mathcal{A}$. By the definition of α-bounded query plans (Section 2), only the upgraded bounded plans consisting of fetch operations with access constraints are to be checked. Hence, it suffices to show the following (upgrading templates to constraints):

> *Under an access schema $\mathcal{A}$ consisting of access constraints only, for any* SPC *query Q, if there exists a chasing sequence that (exactly) covers every tuple in $T(Q)$, then $\xi_Q = (\xi_F, \xi_E)$ is a query plan for Q under $\mathcal{A}$, where ξ_E is the algebra expression of Q,* i.e., *for all datasets $D \models \mathcal{A}$, $\xi_Q(D) = Q(D)$.*

We prove this as follows. (1) We first translate chasing sequences with access constraints only into fetching plans, which, together with evaluation plans, constitute boundedly evaluable query plans under $\mathcal{A}$. This also shows how line 2 of $\mathsf{BEAS}_{\mathsf{SPC}}$ can be implemented. (2) We then (equivalently) interpret the translated query plans ξ in terms of a tableau query $T(Q_\xi)$, referred to as *representing query* of ξ. (3) We finally show that $T(Q_\xi) \equiv Q$ when all tuples in $T(Q)$ are covered by the chasing sequence that derives ξ in step (1).

(1) Fetching plans from chasing sequences. Refer to the chasing rules given earlier for making exactly covered variables and tuples templates as rules (1) and (2), respectively. Since we only need to consider $\mathcal{A}$ with access constraints, we focus on chasing sequences that are deduced by rules (1) and (2) only. We group together *w.l.o.g.* steps in the chasing sequence that share the same "inputs", *i.e.,* the set $t[X]$ of constants and covered variables and access constraint $\gamma = R(X \to Y, N, \bar{0})$ used for deduction via chasing rule (1). That is, when variables y and y' are deduced by the same $t[X]$ and γ via rule (1) (*i.e.,* when both y and y' are in $t[Y]$), we mark both y and y' in the same chase step. For a chasing sequence $\ell = s_1, \ldots, s_n$ like this, we construct a fetching plan ξ^F_ℓ by translating each step s_i into a fetch operation as follows.

- For each chase step s_i ($i \in [1, n]$) in ℓ, if s_i marks variables in t over R by chasing rule (1) or it marks tuple template t by chasing rule (2), via access constraint $\psi = R(X \to Y, N, \bar{0})$, then s_i is translated to $\mathsf{fetch}(X \in \xi_{t[X]}, R, Y, \psi)$, where $\xi_{t[X]}$ is a combination of constants and variables fetched by sub-plans of ξ^F_ℓ, possibly via projection and Cartesian-product.

Therefore, for each chasing sequence ℓ, we can derive a fetching plan ξ^F_ℓ from ℓ via the translation. We next show that (ξ^F_ℓ, ξ_E) is a query plan for Q under $\mathcal{A}$ via (2) and (3).

(2) Fetching plans ξ_ℓ as representing queries $T(Q_\xi)$. We interpret $\xi_\ell = (\xi^F_\ell, \xi_E)$ as a tableau query $T(Q_\xi)$, referred to as the *representing query* of Q. As will be shown in step (3), $T(Q_\xi)$ indeed encodes the query plan ξ_E.

We construct $T(Q_\xi)$ from ξ_ℓ as follows.

(a) Initially, $T(Q_\xi)$ is $T(Q)$.

(b) For each operation $\mathsf{fetch}(X \in \xi_X, R, Y, \varphi)$ in ξ^F_ℓ that is translated from a single step that marks t (or variables in t) of $T(Q)$, add a new tuple t' to $T(Q_\xi)$ such that $t'[XY] = t[XY]$, and for each attribute $A \notin XY$, $t'[A]$ is a new variable that does not appear anywhere else.

(c) For each t in $T(Q_\xi)$, if a tuple template t' that shares the same constants and nontrivial variables as t was added in step (b), then remove t from $T(Q_\xi)$.

Intuitively, all new tuples t' added to $T(Q_\xi)$ in step (b) are to encode the fetching plan ξ_ℓ. Indeed, a $\mathsf{fetch}(X \in T[X], R, Y, \varphi)$ operation is essentially a natural join $T[X] \bowtie I_\varphi[XY]$ on all $D \models \mathcal{A}$, which is encoded by the newly added tuple t' for the fetch in step (b). Step (c) is just to remove

redundant tuples templates from $T(Q_\xi)$ to minimize it. Therefore, (ξ_ℓ^F, ξ_E) is encoded by $(T(Q_\xi), u(Q))$ over all $D \models \mathcal{A}$.

(3) Equivalence between Q_ξ and Q. We next show that $Q_\xi \equiv Q$ (via the conventional notion of query equivalence [6]).

(i) We first show that $Q_\xi \sqsubseteq Q$, *i.e.,* for any D, $Q_\xi(D) \subseteq Q(D)$. Observe that all tuples in $T(Q)$ are removed in step (c) of (2). By the construction of new tuples in step (b) of (2), there exists a homomorphism ρ from $(T(Q), u(Q))$ to $(T(Q_\xi), u(Q))$ such that for each $t \in T(Q)$, $\rho(t[A]) = t'[A]$ for each attribute A, where t' is the tuple corresponding to t (see step (c) of (2)). Note that ρ is a homomorphism from $(T(Q), u(Q))$ to $(T(Q_\xi), u(Q))$ since every t in $T(Q)$ corresponds to a tuple t' in $T(Q_\xi)$. Thus $Q_\xi \sqsubseteq Q$ by Homomorphism Theorem [6].

(ii) Similarly, we show that $Q \sqsubseteq Q_\xi$ by constructing a homomorphism ρ' from $(T(Q_\xi), u(Q))$ to $(T(Q), u(Q))$ as follows. For each tuple t' added in step (b) of (2) above, we let $\rho'(t'[A]) = t[A]$ for each attribute A. Clearly ρ' is a homomorphism from $(T(Q_\xi), u(Q))$ to $(T(Q), u(Q))$ since $T(Q_\xi)$ consists of such new tuples t' only when all tuples t in $T(Q)$ are covered by the chasing sequence ℓ. Thus $Q \sqsubseteq Q_\xi$.

Putting these together, we have that $\xi_\ell \equiv_{\mathcal{A}} Q_\xi \equiv Q$.

(B) Complexity of $\mathsf{BEAS}_{\mathsf{SPC}}$. Observe the following (see Fig. 4). (1) Line 1 of $\mathsf{BEAS}_{\mathsf{SPC}}$ is in $O(|Q||\mathcal{A}_0|)$ time, where $|\mathcal{A}_0|$ contains access constraints in $\mathcal{A}$ only. (2) Lines 2-3 are also in $O(|Q||\mathcal{A}_0|)$ time as they are linear in the size of the fetching plan (bounded by $O(|Q||\mathcal{A}_0|)$) and $|Q|$. (3) line 4 is bounded by $\mathsf{min}(\|\mathcal{A}\|, \|Q\| \log_2 \alpha|D|) * |Q|$ time since the total number of template replacements is bounded by $\mathsf{min}(\|\mathcal{A}\|, \|Q\| \log_2 \alpha|D|)$ and each replacement takes $O(|Q|)$ time. Since $\|\mathcal{A}_0\| \leqslant \|\mathcal{A}\|$ and $\|\mathcal{A}_0\| \leqslant \|Q\| \log_2 \alpha|D|$, $\mathsf{BEAS}_{\mathsf{SPC}}$ can be carried out in $O(|Q|(\mathsf{min}(\|\mathcal{A}\|, \|Q\| \log_2 \alpha|D|)))$-time.

(C) Accuracy analysis of $\mathsf{BEAS}_{\mathsf{SPC}}$. We verify the performance guarantees in three parts.

(1) We first show that $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D) \geqslant \eta$ and $\mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), Q, D) \geqslant \eta$. It suffices to show that $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D) \geqslant \frac{1}{1+d_{\mathsf{rel}}(Q)}$ and $\mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), Q, D) \geqslant \frac{1}{1+d_{\mathsf{cov}}(Q)}$. We show this by induction on the structure of query Q.

Basis. When Q is a relation atom R. Assume that R is retrieved in ξ_α by a fetch operation by using an access template $R(\varnothing \to \mathsf{attr}(R), N, \bar{d})$. By ξ_α and the definition of the lower bound function L, we have that $d_{\mathsf{rel}}(Q) = 0$ and $d_{\mathsf{cov}}(Q) = \sum_{A \in \mathsf{attr}(R)} \bar{d}[A]$. By ξ_α and the definition of RC-measure, $\mathsf{max}_{s \in \xi_\alpha(D)} \mathsf{min}_{r \geqslant 0} \mathsf{max}(r, \mathsf{min}_{t \in Q_r(D)} d(s,t)) = 0 = d_{\mathsf{cov}}(Q)$ and $\mathsf{max}_{t \in Q(D)} \mathsf{min}_{s \in \xi_\alpha(D)} d(s,t) \leqslant \sum_{A \in \mathsf{attr}(R)} \bar{d}[A] = d_{\mathsf{rel}}(Q)$. Therefore, $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D) \geqslant \frac{1}{1+d_{\mathsf{rel}}(Q)}$ and $\mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), Q, D) \geqslant \frac{1}{1+d_{\mathsf{cov}}(Q)}$.

Induction steps. We next consider more complex Q.

(a) When $Q = \sigma_{R[A]=c}(Q')$. Assume that $R[A]$ is retrieved in ξ_α by a fetch operation with access template $\psi = R(X \to Y, N, \bar{d})$ in which $A \in Y$. By the construction of ξ_α in $\mathsf{BEAS}_{\mathsf{SPC}}$ and the definition of L, we have that $d_{\mathsf{rel}}(Q) = d_{\mathsf{rel}}(Q') + \bar{d}[A]$ and $d_{\mathsf{cov}}(Q) = d_{\mathsf{cov}}(Q')$. By ξ_α and the definition of RC-measure, we have that $\mathsf{max}_{s \in \xi_\alpha(D)} \mathsf{min}_{r \geqslant 0} \mathsf{max}(r, \mathsf{min}_{t \in Q_r(D)} d(s,t)) \leqslant \mathsf{max}_{s \in \xi'_\alpha(D)} \mathsf{min}_{r \geqslant 0} \mathsf{max}(r, \mathsf{min}_{t \in Q'_r(D)} d(s,t)) + \bar{d}[A]$, where ξ'_α is the sub-plan of ξ_α that corresponds to sub-query Q' of Q, *i.e.,* ξ_α and ξ'_α share the same fetching plan, but the evaluation plan in ξ'_α only contains operations in the evaluation plan of ξ_α that are relevant to Q'. By the induction hypothesis, $\mathsf{max}_{s \in \xi'_\alpha(D)} \mathsf{min}_{r \geqslant 0} \mathsf{max}(r, \mathsf{min}_{t \in Q'_r(D)} d(s,t)) \leqslant d_{\mathsf{rel}}(Q')$, thus $\mathsf{max}_{s \in \xi_\alpha(D)} \mathsf{min}_{r \geqslant 0} \mathsf{max}(r, \mathsf{min}_{t \in Q_r(D)} d(s,t)) \leqslant d_{\mathsf{rel}}(Q)$. Furthermore, by the chasing rules for constructing the fetching plan ξ_F and the evaluation plan ξ_E for ξ_α, we have that $\mathsf{max}_{t \in Q(D)} \mathsf{min}_{s \in \xi_\alpha(D)} d(s,t) \leqslant d_{\mathsf{cov}}(Q')$. Therefore, by the definition of function L, $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D) \geqslant \frac{1}{1+d_{\mathsf{rel}}(Q)}$ and $\mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), Q, D) \geqslant \frac{1}{1+d_{\mathsf{cov}}(Q)}$.

Similarly one can verify the statement for the case when Q is $\sigma_{R[A] \leqslant c}(Q')$, $\sigma_{R[A] \geqslant c}(Q')$ or $\sigma_{S[A]=S'[B}(Q')$.

(b) When $Q = Q_1 \times Q_2$. Let ξ_1 and ξ_2 be the part of sub-plans of ξ_α that correspond to Q_1 and Q_2, respectively. Then by the construction of ξ_α and the definition of RC-measure, we have that $\mathsf{max}_{s \in \xi_\alpha(D)} \mathsf{min}_{r \geqslant 0} \mathsf{max}(r, \mathsf{min}_{t \in Q_r(D)} d(s,t)) \leqslant \mathsf{max}_{s \in \xi_1(D)} \mathsf{min}_{r \geqslant 0} \mathsf{max}(r, \mathsf{min}_{t \in Q^1_r(D)} d(s,t)) + \mathsf{max}_{s \in \xi_2(D)} \mathsf{min}_{r \geqslant 0} \mathsf{max}(r, \mathsf{min}_{t \in Q^2_r(D)} d(s,t))$, where Q^1_r and Q^2_r are the relaxed queries of Q_1 and Q_2 with r, respectively. By the induction hypothesis, $\mathsf{max}_{s \in \xi_\alpha(D)} \mathsf{min}_{r \geqslant 0} \mathsf{max}(r, \mathsf{min}_{t \in Q_r(D)} d(s,t)) d_{\mathsf{rel}}(Q_1) + d_{\mathsf{rel}}(Q_2) \leqslant d_{\mathsf{rel}}(Q)$. Similarly, by the construction of ξ_α (*e.g.,* the chasing rules and the evaluation plan), $\mathsf{max}_{t \in Q(D)} \mathsf{min}_{s \in \xi_\alpha(D)} d(s,t) \leqslant d_{\mathsf{cov}}(Q_1) + d_{\mathsf{cov}}(Q_2) = d_{\mathsf{cov}}(Q)$. Therefore, $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D) \geqslant \frac{1}{1+d_{\mathsf{rel}}(Q)}$ and $\mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), Q, D) \geqslant \frac{1}{1+d_{\mathsf{cov}}(Q)}$.

Similarly, one can verify the case when $Q = \pi_Y(Q')$.

These verify Theorem 6(1).

(2) We next show the lower bound for η. Observe the following. (a) There exists at least one fetch operation in ξ_α with access template with cardinality $k^* + 1$. Because of this, (b) by the greedy strategy in step (2) of algorithm $\mathsf{BEAS}_{\mathsf{SPC}}$ for choosing access templates, the resolutions on all fetched attributes must be no smaller than $d_0 = \mathsf{max}_{\varphi \in \mathcal{A}_t} \bar{d}^m_{(\varphi, k^*)}$. Thus the relevance distance on $d_{\mathsf{rel}}(Q)$ of L introduced by every fetch operation in ξ_α is no larger than d_0 by the definition of L. Therefore, (c) by the definition of relevance distance d_{rel} of the lower bound function L, $d_{\mathsf{rel}}(Q) \leqslant (\mu_C(Q) + 2\mu_v(Q))d_0 = d^*_{\mathsf{rel}}$. Similarly, $d_{\mathsf{cov}}(Q) \leqslant \sum_{A \in R_Q} d_0 \leqslant d^*_{\mathsf{cov}}$. Hence, we can conclude that $\eta \geqslant \frac{1}{1+\mathsf{max}(d^*_{\mathsf{rel}}, d^*_{\mathsf{cov}})}$.

This verifies Theorem 6(2).

(3) To see that if $\alpha_1 \geqslant \alpha_2$ then $\eta_1 \geqslant \eta_2$, observe that when α grows, by the greedy strategy of step (2) of $\mathsf{BEAS}_{\mathsf{SPC}}$, the resolution of access templates used by all fetch operations in ξ_α does not decrease. Therefore, $d_{\mathsf{cov}}(Q)$ and $d_{\mathsf{rel}}(Q)$ do not increase when α grows, *i.e.,* $\eta_1 \geqslant \eta_2$ if $\alpha_1 \geqslant \alpha_2$.

This verifies Theorem 6(3). □

6. Approximating RA Queries

We continue with the proof of Theorem 1(2) by providing a resource-bounded approximation scheme for RA. The challenge arises from set difference, to ensure that for any approximate answer s returned for RA queries $Q_1 - Q_2$, $s \notin Q_2(D)$ for any dataset D, *without scanning the entire D.*

Approximation scheme $\mathsf{BEAS}_{\mathsf{RA}}$. The scheme, denoted by $\mathsf{BEAS}_{\mathsf{RA}}$, is shown in Fig. 6. It consists of two steps: (1) generate an α-bounded plan ξ_α for Q under $\mathcal{A}$ in D; and (2) compute a lower bound for the accuracy of $\xi_\alpha(D)$. We next outline these steps, highlighting the difference from $\mathsf{BEAS}_{\mathsf{SPC}}$.

Step (1): generating α-bounded plan $\xi_\alpha = (\xi_F, \xi_E)$. One can verify that Lemma 4 holds for RA. Hence $\mathsf{BEAS}_{\mathsf{RA}}$

Algorithm $\mathsf{BEAS_{RA}}$

Input: RA query Q access schema $\mathcal{A} \supseteq \mathcal{A}_t$, and $B = \alpha|D|$.
Output: An α-bounded plan ξ_α for Q under $\mathcal{A}$ and bound η.

1. generate an initial fetching plan ξ_F for Q under $\mathcal{A}$;
2. $\xi_F^\alpha := \mathsf{chAT}(\xi_F, Q, B, \mathcal{A})$;
   /*let d_r^α and d_c^α be the values of d_{rel} and d_{cov} in $L(\xi_F)$ */
3. generate evaluation plan ξ_E for Q *w.r.t.* ξ_F^α; /*let $\xi_\alpha = (\xi_F^\alpha, \xi_E)$ */
4. compute maximal induced $\widehat{Q}$ of Q, an SPC query;
5. generate α-bounded plan $\widehat{\xi}_\alpha$ for $\widehat{Q}$ using $\mathsf{BEAS_{SPC}}$ *w.r.t.* ξ_F;
6. compute distance $d_{\mathsf{cov}}(\widehat{Q})$ in $L(\xi_F^\alpha)$; denote its value as $\widehat{d_c^\alpha}$;
7. $S := \xi_\alpha(D)$; $\widehat{S} := \widehat{\xi_\alpha}(D)$; /* executing ξ_α and $\widehat{\xi_\alpha}$ */
8. $d' := \mathsf{max}_{t \in S'} \mathsf{min}_{s \in S} \delta_{\mathsf{cov}}(\widehat{Q}, t, s)$;
9. $\eta' := \dfrac{1}{1 + \mathsf{max}(d_r^\alpha, d' + \widehat{d_c^\alpha})}$;
10. **return** (ξ_α, η');

Figure 6: Algorithm $\mathsf{BEAS_{RA}}$

also generates canonical bounded plans for RA queries.

Generating initial fetching plan ξ_F. Similar to $\mathsf{BEAS_{SPC}}$, $\mathsf{BEAS_{RA}}$ first generates an initial α-bounded fetching plan ξ_F for Q (line 1), to fetch data for max SPC sub-queries of Q. A *max* SPC *sub-query* of Q is a sub-query Q_s of Q such that (a) Q_s is in SPC, and (b) there exists no SPC sub-query Q'_s of Q such that Q_s is a sub-query of Q'_s. $\mathsf{BEAS_{RA}}$ generates fetching plans for all max SPC sub-queries of Q via $\mathsf{BEAS_{SPC}}$, and groups these plans together to make ξ_F for Q.

Intuitively, a fetching plan for all Q_s's of Q suffices to retrieve all the data needed to compute $Q(D)$.

Optimizing fetching plan. $\mathsf{BEAS_{RA}}$ then chooses access templates with higher resolution for ξ_F via chAT (step (2) of $\mathsf{BEAS_{SPC}}$), and generates an optimized fetching plan ξ_F^α (line 2). Here function L used by chAT is extended to set difference in Q, by letting $d_{\mathsf{rel}}(Q) = d_{\mathsf{rel}}(Q_1)$ and $d_{\mathsf{cov}}(Q) = d_{\mathsf{cov}}(Q_1)$ when $Q = Q_1 - Q_2$; similarly for union $Q_1 \cup Q_2$.

Generating evaluation plan ξ_E. Given ξ_F^α, $\mathsf{BEAS_{RA}}$ generates ξ_E to evaluate Q, and thus completes an α-bounded plan $\xi_\alpha = (\xi_F^\alpha, \xi_E)$ for Q (line 3). To enforce the semantics of set difference, given ξ_F^α, it "implements" an RA query $\mathsf{E}(Q)$, defined inductively based on the structure of Q as follows.

(1) When Q is R, $\sigma_C(Q')$, $Q_1 \times Q_2$, $\pi_Y(Q')$ or $Q_1 \cup Q_2$, $\mathsf{E}(Q)$ is the same as the evaluation plan ξ_E for SPC in $\mathsf{BEAS_{SPC}}$.

(2) When Q is $Q_1 - Q_2$, if relations in Q_2 are fetched with constraints, *i.e.,* templates with $\bar{d} = 0$, then $\mathsf{E}(Q) = \mathsf{E}(Q_1) - \mathsf{E}(Q_2)$. Otherwise $\mathsf{E}(Q) = \mathsf{E}(Q_1) - \pi_{R_{Q_1}} \sigma_C(\mathsf{E}(Q_1) \times \mathsf{E}(\widehat{Q}_2))$. It enforces set difference by expanding Q_2 to its maximal induced query $\widehat{Q}_2$, and by adding a selection condition C.

The *maximal induced query* $\widehat{Q}$ of Q "expands" Q by dropping the negated part of set-differences in Q, such that $\widehat{Q}(D) \supseteq Q(D)$ for all D. The *selection condition* C in $\mathsf{E}(Q)$ identifies and excludes answers to $\mathsf{E}(Q_1)$ that are within a "dangerous" distance $\delta(A)$ to $\mathsf{E}(\widehat{Q}_2)$ on each attribute A involved, such that for any $D \models \mathcal{A}$, if $t \in Q_2(D)$ then t must be in $\pi_{R_{Q_1}} \sigma_C(\mathsf{E}(Q_1) \times \mathsf{E}(\widehat{Q}_2))(D)$.

We next define query $\widehat{Q}_2$. We first present two notions, and then formulate condition C.

(a) A *positive induced query* Q' of Q "expands" Q by dropping the negated part. More specifically, Q' is a query such that (i) its query tree $T_{Q'}$ [6] is a sub-tree of the query tree T_Q of Q; and (ii) for each node v in T_Q labeled with set-difference, the right child of v is *not* included in $T_{Q'}$.

(b) A *maximal induced query* of Q, denoted by $\widehat{Q}$, is a positive induced query of Q such that for any other positive induced query Q' of Q, $T_{Q'}$ is a sub-tree of $T_{\widehat{Q}}$. We take $\widehat{Q}_2$.

We now define the *selection condition* C in $\mathsf{E}(Q)$. It identifies and excludes approximate answers to $\mathsf{E}(Q_1)$ that possibly fall in answers to $\mathsf{E}(\widehat{Q}_2)$, to enforce the semantics of $Q_1 - Q_2$. More specifically, let R_Q be the output schema of Q (similarly, R_{Q_1} and R_{Q_2} for Q_1 and Q_2, respectively). Define $\delta(A) = 0$ if $R_{Q_2}[A]$ is fetched via an access constraint in $\mathcal{A}$ for $\widehat{Q}_2$; and $\delta(A) = \bar{d}_\psi[A]$ if it is fetched via an access template $\psi = R(X \to Y, N, \bar{d}_\psi)$ with $A \in Y$. Then

$$C = \bigwedge_{A \in R_Q} (|\mathsf{dis}_A(R_{Q_1}[A], R_{Q_2}[A])| \leqslant \delta(A)).$$

Example 8: Consider RA query $Q_4 = \pi_B(R(1, B) - W(1, B))$, where a set T_R of tuples is retrieved from R by $\mathsf{fetch}(A \in R, B, \psi_{k_R})$ with template $R(A \to B, 2^{k_R}, \bar{d}_{k_R})$; similarly T_W for W. If we simply compute $T_R - T_W$, we may end up with $(1, b)$ that is in W although it is not in T_W, since T_w cannot cover all tuples of W without scanning the entire dataset.

To enforce the semantics of set difference, $\mathsf{BEAS_{RA}}$ sets $\mathsf{E}(Q_4) = \pi_B(R(1, B) - \pi_{R[AB]}\sigma_C(R(1, B) \times W(1, B)))$, where $C = |\mathsf{dis}_B(R[B], W[B])| \leqslant \bar{d}_{k_W}[B]$. This excludes tuples fetched for $R(1, B)$ that are in $W(1, B)$ via the maximum induced query of $W(1, B)$ (which is itself), since they are within distance $\bar{d}_{k_W}[B]$ of some tuples fetched for S. □

Step (2): computing accuracy bound η for $\xi_\alpha(D)$. In contrast to $\mathsf{BEAS_{SPC}}$ for SPC, L may not serve as an accurate lower bound on the coverage of $\xi_\alpha(D)$ due to the "loss" of approximate answers to Q_1 in $Q = Q_1 - Q_2$. To rectify this, $\mathsf{BEAS_{RA}}$ does the following (lines 4-9 of Fig. 6).

(a) It computes the maximal induced query $\widehat{Q}$ of Q (line 4). It then generates a query plan $\widehat{\xi_\alpha}$ for $\widehat{Q}$ by using part of ξ_F^α for Q relevant to $\widehat{Q}$ as its fetching plan, and the evaluation plan for $\widehat{Q}$ as in $\mathsf{BEAS_{SPC}}$ (line 5). It computes the upper bound for $d_{\mathsf{cov}}(\widehat{Q})$ in L by using the templates in $\widehat{\xi_F^\alpha}$; the value is denoted by $\widehat{d_c^\alpha}$ (line 6). It will use the value to rectify the coverage distance bound for ξ_α.

(b) Let $S = \xi_\alpha(D)$ and $\widehat{S} = \widehat{\xi_\alpha}(D)$. It computes $d' = \mathsf{max}_{t \in \widehat{S}} \mathsf{min}_{s \in S} \delta_{\mathsf{cov}}(\widehat{Q}, t, s)$, the "coverage distance" between tuples in S and S' for $\widehat{Q}$, measuring the "gap" (lines 7-8).

(c) It treats $d' + \widehat{d_c^\alpha}$ as the rectified coverage distance for S, and uses accuracy bound $\eta' = \frac{1}{1 + \mathsf{max}(d_r^\alpha, d' + \widehat{d_c^\alpha})}$ for S (line 9). Here d_r^α is the value of the relevance distance bound $d_{\mathsf{rel}}(\widehat{Q})$ in L with access templates in ξ_F chosen by chAT (line 2).

Intuitively, $\widehat{Q}(D)$ is covered by $\widehat{\xi_\alpha}(D)$ with distance $\widehat{d_c^\alpha}$; and $\widehat{\xi_\alpha}(D)$ is covered by $\xi_\alpha(D)$ with distance d'. Since $Q(D) \subseteq \widehat{Q}(D)$, by the triangle inequality of distance functions, $Q(D)$ is covered by $\xi_\alpha(D)$ with distance $d' + \widehat{d_c^\alpha}$.

Example 9: Recall query Q_4 from Example 8. Assume that approximate answers to R and W are $\{t_1 = (1,1), t_2 = (1,100)\}$ and $\{t_3 = (1,99)\}$, retrieved by $\mathsf{fetch}(A \in R, B, \psi_{k_R})$ and $\mathsf{fetch}(A \in W, B, \psi_{k_W})$, respectively. By the definition of L, d_c^α (see $\mathsf{BEAS_{RA}}$) is $\bar{d}_{k_R}[B]$ such that for any tuple t' in instance of R with $t'[A] = 1$, $\mathsf{min}(|\mathsf{dis}_B(1, t'[B])|, |\mathsf{dis}_B(100, t'[B])|) \leqslant \bar{d}_{k_R}[B]$. By $\mathsf{E}(Q_4)$, assume that t_2 is removed from the answers since t_2 and t_3 are too "close". Then d_c^α is not

an upper bound on the coverage distance $\delta_{\mathsf{cov}}(Q_4, t, s)$.

To rectify this, $\mathsf{BEAS}_{\mathsf{RA}}$ computes upper bound $\widehat{d_c^\alpha}$ on coverage distance of $\widehat{Q}_4$. It also computes the "coverage distance" d' for using t_1 alone to cover $\{t_1, t_2\}$. It then treats $\widehat{d_c^\alpha} + d'$ as an upper bound for $\delta_{\mathsf{cov}}(Q_4, t, s)$, which yields a lower bound on the coverage of the final answers. □

Analysis. With these, $\mathsf{BEAS}_{\mathsf{RA}}$ guarantees the following.

Theorem 7: *For any* RA Q*, access schema* $\mathcal{A} \supseteq \mathcal{A}_t$*, resource ratio* α *and dataset* $D \models \mathcal{A}$*,* $\mathsf{BEAS}_{\mathsf{RA}}$ *generates an* α*-bounded plan* ξ_α *for* Q *under* $\mathcal{A}$ *in* $O(|Q|\mathsf{min}(\|\mathcal{A}\|, \|Q\| \log \alpha|D|))$*-time without accessing* D*, and computes a bound* η *such that*

(1) $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D) \geqslant \eta$ *and* $\mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), Q, D) \geqslant \eta$*;*

(2) $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D) \geqslant \frac{1}{1+(\mu_C(\widehat{Q})+2\mu_v(\widehat{Q}))\cdot \mathsf{max}_{\psi\in\mathcal{A}_0} \bar{d}^m_{(\psi,k^*)}}$*, in which* $k^* = \lfloor \log_2 \frac{\alpha|D|}{\|Q\|} \rfloor - 1$*;*

(3) $\eta > 0$ *when* $\xi_\alpha(D) \neq \varnothing$*;*

(4) if $\alpha_1 \geqslant \alpha_2$*, then* $\eta_1 \geqslant \eta_2$*; and*

(5) if $Q = Q_1 - Q_2$ *and* $t \in Q_2(D)$*, then* $t \notin \xi_\alpha(D)$*,*

where μ_C*,* μ_v*,* $\bar{d}^m_{(\psi,k^*)}$*,* α_i*, and* η_i $(i \in \{1, 2\})$ *are the same as their counterparts in Theorem 6.* □

$\mathsf{BEAS}_{\mathsf{RA}}$ guarantees the following. (1) It returns η above 0, and answers with accuracy at least η. Moreover, the larger α is, the better accuracy is. This verifies Theorem 1 for RA. (2) The lower bound depends on Q, by employing dynamic reduction. Moreover, for the same reason as for $\mathsf{BEAS}_{\mathsf{SPC}}$, the actual bound η and the accuracy of $\xi_\alpha(D)$ are much better than the lower bound estimated. (3) The set difference semantics is strictly enforced, by accessing $\alpha|D|$ tuples only.

We next prove Theorem 7.

Proof: We show the statements of Theorem 7 one by one.

(A) Correctness of $\mathsf{BEAS}_{\mathsf{RA}}$. To show that $\mathsf{BEAS}_{\mathsf{RA}}$ correctly returns bounded plans for Q under $\mathcal{A}$ that enforce set difference of Q, we show the following:

(1) Under any access schema $\mathcal{A} \supseteq \mathcal{A}_t$ of Theorem 1, every RA query Q has a canonical bounded query plan.

(2) For any access schema $\mathcal{A}$, RA $Q = Q_1 - Q_2$ and $D \models \mathcal{A}$, if $t \in Q_2(D)$, then for any fetching plan ξ_F for Q under $\mathcal{A}$, $t \notin \xi_Q(D)$, where $\xi_Q = (\xi_F, E(Q))$ and $E(Q)$ is the evaluation plan for Q *w.r.t.* ξ_F.

From (1) one can show that $\mathsf{BEAS}_{\mathsf{RA}}$ always returns bounded plans for Q under $\mathcal{A}$, along the same lines as the correctness proof of $\mathsf{BEAS}_{\mathsf{SPC}}$. By (2), we know that $\mathsf{BEAS}_{\mathsf{RA}}$ does enforce the semantics of set-difference in Q.

Below we prove (1) and (2).

Proof of (1). For any RA query Q, a canonical query plan ξ_Q for Q can be derived from the group of canonical query plans for all max SPC sub-queries of Q, which are warranted to exist by Lemma 4. It is of the following form: (a) the fetching plan ξ_Q^F is the group of the fetching plans for all max SPC sub-queries of Q; and (b) the evaluation plan is the group of evaluation plans for all max SPC sub-queries of Q, followed by a revision of the original query Q in which each of its max SPC sub-query Q_s is simply replaced by the output relation of the evaluation plan of Q_s. Since every max SPC sub-query of Q has a canonical query plan, this is a canonical query plan for Q as well.

Proof of (2). By the definition of coverage distance, we know that for any α-bounded plans ξ_α returned by $\mathsf{BEAS}_{\mathsf{RA}}$ for Q under $\mathcal{A}$, for any $t \in Q_2(D)$, since $t \in \widehat{Q_2}(D)$, there must exist $s \in \mathsf{E}(\widehat{Q_2})(D_{\xi_F})$ such that $|\mathsf{dis}_A(t[A], s[A])| \leqslant d(A)$, where D_{ξ_F} is the fetched fraction of data from D by the fetching plan ξ_F in ξ_α *i.e.,* s "covers" t. Thus, $\pi_{R_{Q_1}} \sigma_C(\mathsf{E}(Q_1) \times \mathsf{E}(\widehat{Q_2}))$ returns all those answers s in D_{ξ_F} to $\mathsf{E}(Q_1)$ that possibly "cover" exact answers t in $\widehat{Q_2}(D)$ (and thus exact answers in $Q_2(D)$). Therefore, for any fetching plan ξ_F for Q under $\mathcal{A}$, $\xi_\alpha(D) = \mathsf{E}(Q_1 - Q_2)(\xi_F(D))$ contains no answers that possibly "cover" exact answers in $\widehat{Q_2}(D)$ (including those exact answers in $Q_2(D)$).

(B) Complexity of generating plan ξ_α in $\mathsf{BEAS}_{\mathsf{RA}}$. Following the complexity analysis of $\mathsf{BEAS}_{\mathsf{SPC}}$, the generation of ξ_α for Q under $\mathcal{A}$ (lines 1-3 of $\mathsf{BEAS}_{\mathsf{RA}}$ (Fig. 6) can be implemented in $O(|Q|(\mathsf{min}(\|\mathcal{A}\|, \|Q\| \log_2 \alpha|D|)))$ time.

(C) Accuracy analysis of $\mathsf{BEAS}_{\mathsf{RA}}$.

Proof of Theorem 7(1). By (A2) in the correctness proof above and the definition of relevance ratio, when $Q = Q_1 - Q_2$, $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D) = \mathsf{F}_{\mathsf{rel}}(\xi_1(D), Q_1, D)$, where ξ_1 is the subplan of ξ_α for Q_1. Hence, along the same liens as the proof of Theorem 6, one can verify that $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D) \geqslant \frac{1}{1+d_r^\alpha} \geqslant \eta$ (see line 2 of $\mathsf{BEAS}_{\mathsf{RA}}$ for d_r^α) and moreover, η is the lower bound returned by $\mathsf{BEAS}_{\mathsf{RA}}$.

As remarked in Section 6, $\frac{1}{1+d_c^\alpha}$ is not necessarily a lower bound for $\mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), Q, D)$ due to the way we enforce set difference semantics in $\mathsf{E}(Q_1 - Q_2)$. Below we show that $\mathsf{BEAS}_{\mathsf{RA}}$ rectifies this, *i.e.,* $\mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), Q, D) \geqslant \eta$.

Lemma 8: *For any dataset* $D \models \mathcal{A}$ *and* RA *query* Q*, consider* ξ_α*,* $\widehat{\xi_\alpha}$*,* $\widehat{d_c^\alpha}$ *and* d' *in algorithm* $\mathsf{BEAS}_{\mathsf{RA}}$ *for* Q *and* $\widehat{Q}$*,*

(a) $Q(D) \subseteq \widehat{Q}(D)$ *and* $\xi_\alpha(D) \subseteq \widehat{\xi_\alpha}(D)$*;*

(b) $\mathsf{F}_{\mathsf{cov}}(\widehat{\xi_\alpha}(D), \widehat{Q}, D) \geqslant \frac{1}{1+\widehat{d_c^\alpha}}$*; and*

(c) $\mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), Q, D) \geqslant \mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), \widehat{Q}, D)$. □

Indeed, (a) is ensured by the definition of maximal induced queries; (b) follows from the proof of Theorem 6 (observe that $\widehat{Q}$ is an SPC); and (c) follows from (a) and the definition of coverage ratio in the RC-measure.

By (b), for any tuple $t \in \widehat{Q}(D)$, there exists a tuple $\widehat{s} \in \widehat{\xi_\alpha}(D)$ such that $\delta_{\mathsf{cov}}(\widehat{Q}, t, \widehat{s}) \leqslant \widehat{d_c^\alpha}$. By line 8 of algorithm $\mathsf{BEAS}_{\mathsf{RA}}$, for any $\widehat{s} \in \widehat{\xi}_*(D)$, there exists $s \in \xi_\alpha(D)$ such that $\delta_{\mathsf{cov}}(\widehat{Q}, \widehat{s}, s) \leqslant d'$. Therefore, by the triangle inequality of the distance functions, for any $t \in \widehat{Q}(D)$, there exists $s \in \xi_\alpha(D)$ such that $\delta_{\mathsf{cov}}(\widehat{Q}, t, s) \leqslant \widehat{d_c^\alpha} + d'$. By the definition of coverage ratio, we have that $\mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), \widehat{Q}, D) \geqslant \frac{1}{1+\widehat{d_c^\alpha}+d'} = \eta$. By (c), $\mathsf{F}_{\mathsf{cov}}(\xi_\alpha(D), Q, D) \geqslant \eta$.

This completes the proof of Theorem 7(1).

Proof of Theorem 7(2). Observe that $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D) \geqslant \mathsf{F}_{\mathsf{rel}}(\widehat{\xi_\alpha}(D), \widehat{Q}, D)$. Since $\widehat{Q}$ is an SPC query, by the proof of Theorem 6, we have that $\mathsf{F}_{\mathsf{rel}}(\widehat{\xi_\alpha}(D), \widehat{Q}, D) \geqslant \frac{1}{1+(\mu_C(\widehat{Q})+2\mu_v(\widehat{Q}))\cdot \mathsf{max}_{\varphi\in\mathcal{A}_0} \bar{d}^m_{(\varphi,k^*)}}$. So is $\mathsf{F}_{\mathsf{rel}}(\xi_\alpha(D), Q, D)$.

Proof of Theorem 7(3). Note that when $\xi_\alpha(D) \neq \varnothing$, $d' \neq +\infty$. Thus $\eta \neq 0$.

Proof of Theorem 7(4). This is verified along the same lines as the proof of Theorem 6(3).

Proof of Theorem 7(5). This is assured by the correctness proof (A2) of $\mathsf{BEAS}_{\mathsf{RA}}$ given above.

This completes the proof of Theorem 7. □

7. Approximating Aggregate Queries

We next extend the approximation scheme to $\mathsf{RA}_{\mathsf{aggr}}$ queries, denoted by $\mathsf{BEAS}_{\mathsf{agg}}$, under access schema $\mathcal{A}$ that subsumes $\mathcal{A}_t$ of Section 4.1, accommodating bag semantics.

Handling max and min. We start with $\mathsf{RA}_{\mathsf{aggr}}$ queries Q defined with max and min. One can verify that Lemma 4 also holds on Q. $\mathsf{BEAS}_{\mathsf{agg}}$ generates canonical bounded plans $\xi_\alpha = (\xi_F, \xi_E)$ for Q. It works in the same way as $\mathsf{BEAS}_{\mathsf{RA}}$ except the following extensions. Let $Q = \mathsf{gpBy}(Q', X, \mathsf{agg}(V))$.

One can verify that Lemma 4 also holds on such $\mathsf{RA}_{\mathsf{aggr}}$ queries defined with max and min. Hence $\mathsf{BEAS}_{\mathsf{agg}}$ also generates canonical bounded plans $\xi_\alpha = (\xi_F, \xi_E)$ for $\mathsf{RA}_{\mathsf{aggr}}$ queries $Q = \mathsf{gpBy}(Q', X, \mathsf{agg}(V))$, similar to RA. It works in the same way as $\mathsf{BEAS}_{\mathsf{RA}}$ for RA, with the following extensions.

Lower bound function L revised. The relevance and coverage upper bounds in L are $d_{\mathsf{rel}}(Q) = d_{\mathsf{rel}}(Q')$ and $d_{\mathsf{cov}}(Q) = d_{\mathsf{cov}}(Q')$. Using the revised L, it generates an α-bounded fetching plan ξ_F via chAT just like in $\mathsf{BEAS}_{\mathsf{RA}}$.

Evaluation plan ξ_E. It defines $\mathsf{E}(Q) = \mathsf{gpBy}(\mathsf{E}(Q'), X, \mathsf{agg}(V))$. The rest remains the same as $\mathsf{BEAS}_{\mathsf{RA}}$ (see Section 6 for $\mathsf{E}(Q)$). That is, the group-by and aggregate functions of Q are executed on approximate answers to Q'.

With such extensions, $\mathsf{BEAS}_{\mathsf{agg}}$ works in the same way as $\mathsf{BEAS}_{\mathsf{RA}}$, as follows: it (a) generates an α-bounded plan ξ_α for Q under $\mathcal{A}$; and (b) computes the accuracy bound η by making use of the maximum induced query $\widehat{Q}$ of Q.

Corollary 9: *For ξ_α and η generated by $\mathsf{BEAS}_{\mathsf{agg}}$, the lower bounds and properties on ξ_α and η specified in Theorem 7 remain the same for any $\mathsf{RA}_{\mathsf{aggr}}$ query with min and max.* □

Proof: By the definition of RC-measure for group-by aggregate queries with min and max, we know that the relevance and coverage ratio are the same as the corresponding RA queries (without group-by aggregation), provided that the **group by** semantics is enforced on the query plans. Since $\mathsf{BEAS}_{\mathsf{agg}}$ enforces the **group by** semantics in terms of the evaluation plan, the accuracy bounds for RA developed in Theorem 7 remain the same for $\mathsf{RA}_{\mathsf{aggr}}$ with min and max. □

For $\mathsf{RA}_{\mathsf{aggr}}$ queries including sum, avg and count, we need to extend access schema to keep track of the number of duplicated attribute values. In a template $R(X \to Y, N, \bar{d}_Y)$, given an X-value $\bar{a}$, its index additionally returns the number of occurrences of each returned Y-value $\bar{b}$, by aggregating over all the Y-values in D "represented" by $\bar{b}$. Under this extension, $\mathsf{BEAS}_{\mathsf{agg}}$ can be extended.

More specifically, we extend access constraints and access templates defined in Section 2 as follows.

Extended access constraints. For each access constraint $\varphi = R(X \to Y, N, \bar{0})$, we extend it as follows:

$$\varphi' = R(X \to Y, N, \bar{0}, \overline{\mathsf{occ}}),$$

where $\overline{\mathsf{occ}}$ is an additional attribute to keep track of the number of occurrences of XY.

A database instance D of relation schema R satisfies ϕ' if

- it satisfies ϕ as before (Section 2); and moreover
- there exists an index on D that, given each X-value $\bar{a}$, for each of the Y-value $\bar{b}$, returns $D_{XY}(X = \bar{a})$ in $O(N)$ time, along with an $\overline{\mathsf{occ}}$ value for $\overline{ab} \in D_{XY}(X = \bar{a})$, recording the occurrences of $\overline{ab}$ in D on attributes XY.

The occurrence count $\overline{\mathsf{occ}}$ allows us to calculate aggregate values when answering $\mathsf{RA}_{\mathsf{aggr}}$ queries.

Extended access templates. Similarly, for each access template $\psi = R(X \to Y, 2^k, \bar{d})$, we extend it to

$$\psi' = R(X \to Y, N, \bar{d}, \overline{\mathsf{occ}}),$$

such that a database instance D satisfies ψ' if

- D satisfies ψ; and
- there exists an index on D that, for each X-value $\bar{a}$, returns N values in $D_{XY}(X = \bar{a})$ with resolution $\bar{d}$ as usual, and an exact occurrence count for each value $\overline{ab}$ returned, which is the sum of occurrences of all values in $D_{XY}(X = \bar{a})$ represented by $\overline{ab}$.

Intuitively, with the extension of occurrence, bounded query plans can enforce the bag semantics of fetched tuples via the occurrences without actually fetching those duplicated values, and can "restore" the bag semantics of intermediate answers by arithmetic calculation over the occurrences when they are needed by the aggregate functions. Therefore, algorithm $\mathsf{BEAS}_{\mathsf{agg}}$ can also work under bag semantics.

8. Experimental Study

Using real-life and synthetic data, we conducted five sets of experiments to evaluate (1) the quality of approximate answers; (2) bound η; (3) resource ratios for exact answers; (4) index size; and (5) the efficiency and scalability of BEAS.

Experimental setting. We used two real-life datasets. (1) AIRCA integrates Flight On-Time Performance data [1] and Carrier Statistic data [2] for US air carriers from 1987 to 2014. It consists of 7 tables, 358 attributes, and over 162 million tuples, about 60GB of data. (2) TFACC is a dataset of road accidents that happened in the UK from 1979 to 2005 [3], and National Public Transport Access Nodes [4]. It has 19 tables with 113 attributes, and over 89.7 million tuples, about 21.4GB of data. We also used (3) synthetic data (TPCH) generated by TPC-H dbgen [5], varying scale factor σ from 5 to 25, about 200 million tuples when $\sigma = 25$.

Access schema. We picked 7, 12 and 9 access constraints for AIRCA, TFACC and TPCH, respectively, from those used in [11, 13]. For each access constraint $R(X \to Y, N, 0)$, we added access templates $R(XY \to Z, 2^i, \bar{d}_i)$ for $i \in [0, \lceil \log_2 \mathsf{max}_{\bar{a} \in D_{XY}} |D_Z(XY = \bar{a})| \rceil]$, where $Z = \mathsf{attr}(R) \setminus XY$. We also included templates in $\mathcal{A}_t$ of Section 4, yielding a total of 617, 573 and 440 access templates, respectively. These constitute 24, 31 and 20 distinct templates when grouped by their X and Y attributes. For these templates, we built their indices as described in Section 4. Sample access constraints and templates can be found in Appendix B.

Queries. We generated 90 queries (30 for each dataset), among which 30% are aggregate SPC queries; the others are RA queries, each with 0-3 set differences. Half of the attributes in the queries are from the access constraints. The queries varied in (a) the number #-sel of predicates in the selection condition σ_C of Q, in the range of [3, 7], and (b) the number #-prod of Cartesian products in Q, in the range of [0, 4]. For AIRCA and TFACC, we drew attributes values for the queries randomly from the datasets. For TPCH, we extended its built-in 22 queries to 30.

Sample queries are provided in Appendix B.

Algorithms. We implemented the following: (1) BEAS for SPC and RA (aggregate or not), denoted by $\mathsf{BEAS}_{\mathsf{SPC}}$ and $\mathsf{BEAS}_{\mathsf{RA}}$, respectively, based on the algorithms of Sections 5, 6 and 7; (2) Sampl, an extension of sampling approximation of [17] that samples $\alpha|D|$ tuples and answers queries using the sample; (3) Histo, which creates multi-dimensional histograms of size $\alpha|D|$ and uses it to answer queries [28].

We also compared BEAS with (4) BlinkDB, which supports aggregate SPC queries (no min/max) with restricted joins [8]. For each access template $R(X \to Y, N, \bar{d}_Y)$, we created a sample for BlinkDB by using "**create** table R_SAMPLE **as select distinct** X, Y **from** R c", where $c \in [1, \frac{N}{\|R[XY]\|}]$, and $\|R[XY]\|$ is the number of tuples in $\pi_{XY}(R)$. As such, we provided BlinkDB with indices of our access schema as samples. Although BlinkDB claims to be able to quantify controls on running time and accessed data [8], we could not configure it following its available document. Thus we manually simulated its stratified sampling strategy [7, 8] and controlled its accessed data via the size of sample tables.

We evaluated each method using all queries it supports: (a) RA for BEAS and Sampl, aggregate or not, (b) SPC for Histo, aggregate or not, and (c) restricted aggregate SPC for BlinkDB. This is in favor of Histo and BlinkDB since RA queries are harder to approximate, while Histo and BEAS can achieve good accuracy on simpler queries. We used full datasets in the experiments unless stated otherwise.

The experiments were conducted on an Amazon EC2 d2.xlarge instance with 4 EC2 compute units, 30.5GB memory and 4TB HDD storage, using PostgreSQL (9.6devel) and MySQL 5.7 as underlying DBMS. All the experiments were run 3 times, and the average is reported here.

Experimental results. We next report our findings.

Exp-1: Accuracy of approximate answers. We evaluated the accuracy of all these methods using three accuracy measures: (a) MAC [28], the measure of Histo (Section 3, normalized to [0, 1]), (b) F-measure, and (c) RC-measure. We did not experiment with the measures of Sampl and BlinkDB since (1) Sampl does not evaluate the accuracy of set-valued answers; and (2) BlinkDB adopts confidence intervals to measure accuracy; it only works on restricted group-by aggregate queries; it requires to access all exact values of the group-by attributes, and is not always possible under a resource ratio α; moreover, it is developed for probabilistic sampling approaches, as opposed to BEAS.

We evaluated the RC measure with full TPCH, TFACC and AIRCA. For MAC, we only used synthetic TPCH to illustrate the impact of $|D|$; the results on the real-life datasets are consistent. We found that F-measure is 0 in all cases for most queries, and hence do not show it in the figures.

(1) Varying α. Varying α from 1.5×10^{-4} to 5.5×10^{-4}, we evaluated its impact on the average accuracy of the methods.

(A) RC-measure. We find the following. As reported in Figures 7(a), 7(b), 7(c), (a) the approximate answers computed by BEAS are accurate. $\mathsf{BEAS}_{\mathsf{SPC}}$ and $\mathsf{BEAS}_{\mathsf{RA}}$ are consistently above 0.72; $\mathsf{BEAS}_{\mathsf{SPC}}$ is above 0.85 when $\alpha \geqslant 3.5 \times 10^{-4}$, 4.5×10^{-4} and 5.5×10^{-4} on TFACC, AIRCA and TPCH, respectively; and $\mathsf{BEAS}_{\mathsf{RA}}$ is above 0.82 in the same setting; (b) BEAS outperforms Sampl, Histo and BlinkDB; *e.g.,* when α is 1.5×10^{-4} on TPCH, on average $\mathsf{BEAS}_{\mathsf{RA}}$ is 24, 3.4 and 2.0 times more accurate than Sampl, Histo and BlinkDB, respectively, and the gap for $\mathsf{BEAS}_{\mathsf{SPC}}$ is larger; the results on the other datasets are similar. (c) The larger α is, the more accurate BEAS is. This is because when BEAS generates α-bounded plans, it can inspect more tuples *guided by input query Q* to pick more relevant tuples. In contrast, Sampl and Histo use one-size-fit-all synopses and are less sensitive to α. While BlinkDB dynamically select samples, it does not work very well on queries with joins, max or min. These verify the *benefit of dynamic data reduction* of BEAS.

(B) MAC measure. As shown in Fig. 7(d), under MAC, (a) $\mathsf{BEAS}_{\mathsf{SPC}}$ and $\mathsf{BEAS}_{\mathsf{RA}}$ also perform better than the others: 18.3, 2.4, and 1.9 times more accurate than Sampl, Histo and BlinkDB on TPCH, respectively, when $\alpha = 1.5 \times 10^{-4}$. Their MAC accuracy is even a bit higher than their RC accuracy. This is because the RC-measure subsumes the MAC measure with coverage $\mathsf{F}_{\mathsf{cov}}$ (Section 3), and additionally assesses relevance. (b) Histo still falls behind $\mathsf{BEAS}_{\mathsf{SPC}}$, $\mathsf{BEAS}_{\mathsf{RA}}$ and BlinkDB, since it uses the same synopsis of size $\alpha|D|$ for all queries, as opposed to BEAS and BlinkDB. Its gap from BlinkDB is smaller than its RC accuracy since Histo is optimized for MAC. (c) The MAC accuracy of BlinkDB is lower than $\mathsf{BEAS}_{\mathsf{SPC}}$ and $\mathsf{BEAS}_{\mathsf{RA}}$, although also a bit higher than its RC accuracy, *e.g.,* when $\alpha \geqslant 4.5 \times 10^{-4}$, BlinkDB is 0.45 while $\mathsf{BEAS}_{\mathsf{RA}}$ and $\mathsf{BEAS}_{\mathsf{SPC}}$ are above 0.73 and 0.86, respectively, for the same reason as under the RC measure.

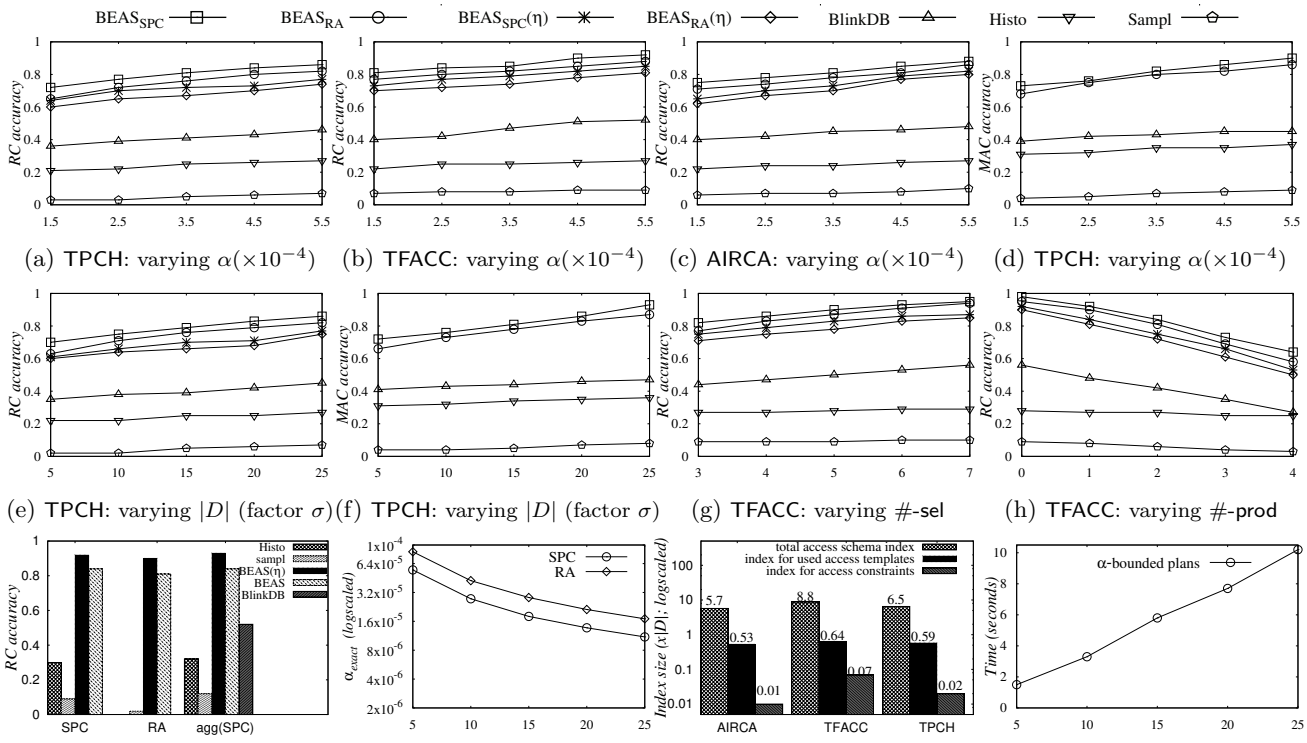
(2) Varying $|D|$. Fixing $\alpha = 5.5 \times 10^{-4}$, we varied the scale factor σ of TPCH from 5 to 25, to assess the impact of $|D|$.

(A) RC-measure. As shown in Fig. 7(e), (a) $\mathsf{BEAS}_{\mathsf{SPC}}$ and $\mathsf{BEAS}_{\mathsf{RA}}$ are above 0.75 all the time. (b) BEAS performs better than Sampl, Histo and BlinkDB. This is more evident on larger datasets: $\mathsf{BEAS}_{\mathsf{SPC}}$ and $\mathsf{BEAS}_{\mathsf{RA}}$ are above 0.82 when $\sigma \geqslant 0.8$. In contrast, Sampl, Histo and BlinkDB are not very sensitive to $|D|$. This is because when D grows, BEAS takes advantage of the larger budget $\alpha|D|$ to find tuples more relevant to query Q, and hence improve S. In contrast, Sampl and Histo do not benefit much from larger D, since their searches are confined to a (larger) predefined synopsis; similarly for BlinkDB due to its restricted sample selection.

(B) MAC measure. As shown in Fig. 7(f), (a) $\mathsf{BEAS}_{\mathsf{SPC}}$ and $\mathsf{BEAS}_{\mathsf{RA}}$ have the highest MAC accuracy among all. Histo and Sampl fall behind, but Histo is closer to BlinkDB than its RC accuracy. (b) BEAS benefits more from larger $|D|$. For instance, when σ varies from 5 to 25, $\mathsf{BEAS}_{\mathsf{SPC}}$ increases from 0.72 to 0.93, while it is from 0.31 to 0.36 for Histo; similarly for BlinkDB. The results are consistent with Fig. 7(d).

(3) Varying Q. Fixing $\alpha = 5.5 \times 10^{-4}$, we varied #-sel, #-prod and the types of queries Q to evaluate the impact of Q. We report the RC accuracy on TFACC in Figures 7(g), 7(h) and 7(i); the results on AIRCA and TPCH are consistent, under either RC or MAC measure (not shown). We treat the accuracy of Histo for RA as 0 since it does not support RA; similarly for BlinkDB on non-aggregate queries.

We find the following. (1) BEAS does better with larger #-sel, since its query plans are guided by relevance to Q (see Section 5). In contrast, Histo and Sampl are indifferent to #-sel since their searches are confined to one-size-fit-all synopses, and do not explore selection conditions to improve accuracy. The accuracy of BlinkDB also benefits from #-sel, but not as much as BEAS. (2) BEAS and BlinkDB perform worse with larger #-prod since Cartesian products scale up



(a) TPCH: varying $\alpha(\times 10^{-4})$ (b) TFACC: varying $\alpha(\times 10^{-4})$ (c) AIRCA: varying $\alpha(\times 10^{-4})$ (d) TPCH: varying $\alpha(\times 10^{-4})$

(e) TPCH: varying $|D|$ (factor σ) (f) TPCH: varying $|D|$ (factor σ) (g) TFACC: varying #-sel (h) TFACC: varying #-prod

(i) TFACC: varying query types (j) TPCH: varying $|D|$ (factor σ) (k) Index size ($\times|D|$) (l) TPCH: varying $|D|$ (factor σ)

Figure 7: Accuracy and scalability of BEAS, the resource-bounded approximation framework

the total distances of values (see Section 3). Histo and Sampl also degrade with larger #-prod, but are less sensitive as their accuracy is dominated by synopses. (3) BEAS outperforms Histo, Sampl and BlinkDB, and does the best for SPC.

(4) Quality of RC*-measure*. We manually examined and verified that RC accurately ranks sensible answers, which are often overlooked by other measures, as shown below.

Example 10: Consider query q_1 posed on TFACC, to *find casualties in accidents that happened at location (52.136436, -0.460739)* (see Appendix B), by joining relations vehicles, naptan and naptantraffic. The exact answer to q_1 is $\varnothing$ since there is no atcocode (bus stop ID) in the naptan relation that exactly matches location (52.136436, -0.460739). Because of this, for any non-empty approximate answers S to q_1 in TFACC, both the F-measure and MAC measure of S is 0.

Nonetheless, BEAS finds a set S' of casualties recorded in TFACC that were involved in accidents near the bus stop with atcocode 20, which is at location (52.136334, -0.462906), close to (52.136436, -0.460739). Obviously S' consists of sensible answers to q_1. The RC accuracy of S' is non-zero, as opposed to the F-measure and MAC measure of S'. □

Exp-2: Accuracy of bounded η. We evaluated the quality of estimated lower bound η, denoted by $\mathsf{BEAS}_{\mathsf{SPC}}(\eta)$ and $\mathsf{BEAS}_{\mathsf{RA}}(\eta)$ for SPC and RA, respectively, aggregate or not.

In the same setting as Exp-1, the results are reported in Figures 7(a), 7(b), 7(c) (varying α), Figure 7(e) (varying $|D|$) and Figures 7(g), 7(h) and 7(i) (varying Q). These verify that the lower bound η estimated by BEAS is close to the accuracy of answers. For instance, when $\mathsf{BEAS}_{\mathsf{SPC}}$ is 0.92 on TFACC with $\alpha = 5.5 \times 10^{-4}$, $\mathsf{BEAS}_{\mathsf{SPC}}(\eta)$ is 0.85. The other methods do not compute an accuracy bound like η.

Exp-3: Resource ratio for exact answers. Varying the scale $|D|$ of TPCH as in Exp-1(2), we evaluated the average α_{exact} for BEAS to find exact answers for queries in Exp-1(2).

As shown in Fig. 7(j), the larger $|D|$ is, the smaller α_{exact} is. On full TPCH, α_{exact} is 2.6×10^{-6} for SPC and 4.1×10^{-6} for RA. Indeed, (a) the majority of queries that are answered exactly in Exp-2(b) are boundedly evaluable [11]. Their plans access a bounded amount of data independent of $|D|$, which is typically very small. (b) For those queries that are not bounded, BEAS applies access constraints as much as possible; the remaining parts amount for only a small part of D for approximation with access templates.

Exp-4: Index size. We also examined the indices of access schema. As reported in Fig. 7(k), in all three datasets, the indices for access constraints take less than 7% of the size $|D|$ of the dataset D in all three cases, and its entire indices (with templates) are in $c|D|$ for $5.7 < c < 8.8$, among which the indices of access schema that are actually used by BEAS for answering queries account for less than 64% of $|D|$.

We remark the following. (1) The compared methods use about the same amount of indices, *e.g.,* BlinkDB also employs all the indices as its samples in order to achieve the accuracy reported. (2) The total index size is comparable to typical indices of traditional DBMS, *e.g.,* 3-5 times of $|D|$ for TPCH [32]. (3) To answer a query, BEAS accesses at most $\alpha|D|$ tuples no matter whether the tuples are from the indices of an access schema or the original D, and no matter how big D and the indices are. (4) It is possible to reduce indices by using "optimal access schema", as BEAS only needs to use a fraction of the simple access schema tested.

Exp-5: Efficiency and scalability. Finally, we evaluated (a) the efficiency of BEAS for generating α-bounded plans; and (b) the scalability of the plans. For all queries, BEAS generates α-bounded plans in less than 200ms. In the same setting as Exp-2(b), we report the scalability of the plans. As

shown in Fig. 7(l), the plans scale well with $|D|$, as expected, as they access only an α-fraction of D. They took at most 10.2 seconds on full datasets, while both PostgreSQL and MySQL could not finish within 3 hours (hence not shown).

Summary. We find the following. (1) The RC-measure quantifies the quality of approximate answers and reflects the changes to resource ratio α, as opposed to, *e.g.,* F-measure. (2) BEAS has accuracy consistently above 0.85 when $\alpha \geqslant 5.5 \times 10^{-4}$ on all three datasets for SPC queries, and above 0.82 for RA, aggregate or not, under both RC-measure and MAC measure. Better still, the RC-measure gets higher when either dataset D or resource ratio α grows larger. (3) BEAS is able to find exact answers for RA queries under α as small as 4.1×10^{-6} on AIRCA. (4) BEAS is efficient and scalable. It generates α-bounded plans in 200ms, and the plans compute answers for all queries within 10.2 seconds for $\alpha = 5.5 \times 10^{-4}$, even on AIRCA of 60GB and TPCH with 200 million tuples, while conventional PostgreSQL and MySQL cannot terminate within 3 hours. (5) The bound η estimated by BEAS is accurate. (6) While Histo, Sampl and BlinkDB are effective when answering simple aggregate queries, they are not as accurate as BEAS for full (aggregate) SPC and RA, especially on queries with multiple selections or joins.

9. Conclusion

We have proposed BEAS, a framework for querying (big) relations with bounded resources. Its novelty consists of access templates, a new accuracy measure, a resource-bounded approximation scheme, and resource-bounded algorithms for answering SPC, RA and $\mathsf{RA}_{\mathsf{aggr}}$ queries with a deterministic accuracy lower bound. Our experimental study has verified that BEAS is promising for answering unpredictable queries, aggregate or not, under a small resource ratio.

The work is a step towards striking a balance between available resources and accuracy. We are currently deploying and evaluating BEAS at sites of our industry collaborators.

10. References

[1] *http://www.transtats.bts.gov/DatabaseInfo.asp?DB_ID=120*.
[2] *http://www.transtats.bts.gov/DatabaseInfo.asp?DB_ID=110*.
[3] *http://data.gov.uk/dataset/road-accidents-safety-data*.
[4] *http://data.gov.uk/dataset/naptan*.
[5] TPC-H. *http://www.tpc.org/tpch/*.
[6] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[7] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *SIGMOD*, 2000.
[8] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: Queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.
[9] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, 2003.
[10] M. Benedikt, J. Leblay, B. ten Cate, and E. Tsamoura. *Generating Plans from Proofs: The Interpolation-based Approach to Query Reformulation*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2016.
[11] Y. Cao and W. Fan. An effective syntax for bounded relational queries. In *SIGMOD*, 2016.
[12] Y. Cao, W. Fan, F. Geerts, and P. Lu. Bounded query rewriting using views. In *PODS*, 2016.
[13] Y. Cao, W. Fan, T. Wo, and W. Yu. Bounded conjunctive queries. *PVLDB*, 2014.
[14] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *VLDB J.*, 10(2-3), 2001.
[15] S. Chaudhuri. Generalization and a framework for query modification. In *ICDE*, 1990.
[16] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, 2005.
[17] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *FTDB*, 4(1–3), 2012.
[18] B. Cule, F. Geerts, and R. Ndindi. Space-bounded query approximation. In *ADBIS*, 2015.
[19] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. Orthogonal range searching: Querying a database. *Computational Geometry: Algorithms and Applications*, pages 95–120, 2008.
[20] B. Ding, S. Huang, S. Chaudhuri, K. Chakrabarto, and C. Wang. Sample + seek: Approximating aggregates with distribution precision guarantee. In *SIGMOD*, 2016.
[21] R. Elmasri. *Fundamentals of database systems*. Pearson Education India, 2008.
[22] Facebook. Introducing Graph Search. *https://en-gb.facebook.com/about/graphsearch*, 2013.
[23] W. Fan, F. Geerts, Y. Cao, and T. Deng. Querying big data by accessing small data. In *PODS*, 2015.
[24] W. Fan, F. Geerts, and L. Libkin. On scale independence for querying big data. In *PODS*, 2014.
[25] W. Fan, X. Wang, and Y. Wu. Distributed graph simulation: Impossibility and possibility. *PVLDB*, 7(12), 2014.
[26] I. Grujic, S. Bogdanovic-Dinic, and L. Stoimenov. Collecting and analyzing data from e-government Facebook pages. In *ICT Innovations*, 2014.
[27] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the Hausdorff distance. In *TPAMI*, 1993.
[28] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *VLDB*, 1999.
[29] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, 2009.
[30] A. Kadlag, A. V. Wanjari, J. Freire, and J. R. Haritsa. Supporting exploratory queries in databases. In *DASFAA*, 2004.
[31] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri, and B. Ding. Quickr: Lazily approximating complex ad-hoc queries in big data clusters. In *SIGMOD*, 2016.
[32] C. Levine. TPC benchmarks. *http://research.microsoft.com /en-us/um/people/gray/WICS_99_TP/22_Levine.ppt*.
[33] C. Li. Computing complete answers to queries in the presence of limited access patterns. *VLDB J.*, 12(3), 2003.
[34] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. Xu. Regret-minimizing representative databases. In *VLDB*, 2010.
[35] A. Nash and B. Ludäscher. Processing first-order queries under limited access patterns. In *PODS*, 2004.
[36] G. Ramalingam and T. Reps. On the computational complexity of dynamic graph problems. *TCS*, 158(1-2), 1996.
[37] R. Rubinfeld and A. Shapira. Sublinear time algorithms. *SIAM Journal on Discrete Mathematics*, 25(4), 2011.
[38] Stack Overflow. SQL: Inner joining two massive tables. *http://stackoverflow.com/questions/1750001/sql-inner-joining-two-massive-tables*.
[39] L. J. Stockmeyer. The polynomial-time hierarchy. *TCS*, 3(1), 1976.
[40] J. T. Thayer, R. Stern, A. Felner, and W. Ruml. Faster bounded-cost search using inadmissible estimates. In *ICAPS*, 2012.
[41] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3), 1996.

APPENDIX A: Definitions

Definition of $\mathsf{tariff}(\xi)$ **(Section 5)**

Given a bounded query plan ξ generated by $\mathsf{BEAS}_{\mathsf{SPC}}$, its $\mathsf{tariff}(\xi)$ of accessed tuples is estimated as follows:

- if $\xi = \{c\}$, then $\mathsf{tariff}(\xi) = 1$, *i.e.,* it accesses only one tuple that is $\{c\}$;
- if $\xi = \sigma_C(\xi')$ or $\pi_Y(\xi')$, then $\mathsf{tariff}(\xi) = \mathsf{tariff}(\xi')$; that is, selection and projection do not access extra tuples by ξ';
- if $\xi = \xi_1 \times \xi_2$, $\xi_1 \cup \xi_2$ or $\xi_1 - \xi_2$, then $\mathsf{tariff}(\xi) = \mathsf{tariff}(\xi_1) + \mathsf{tariff}(\xi_2)$; that is, these binary algebra operators do not access extra tuples from the underlying dataset;
- if $\xi = (T = \xi', \mathsf{fetch}(X \in T, R, Y, \psi))$ in which $\psi = R(X \to Y, N_\psi, \bar{d}_\psi)$, then $\mathsf{tariff}(\xi) = \mathsf{tariff}(\xi') * N_\psi$. Indeed, the size $|T|$ of T, *i.e.,* the output of ξ', is bounded by $\mathsf{tariff}(\xi')$. Therefore, ξ accesses no more than $|T| * N_\psi \leqslant \mathsf{tariff}(\xi') * N_\psi$ tuples.

Intuitively, for any $D \models \mathcal{A}$, $\xi(D)$ access no more than $\mathsf{tariff}(\xi)$ tuples.

APPENDIX B: Experiment Details

Access schema used in the experiments

(A) Access constraints.

(1) TFACC .

$\psi_1 = \mathsf{accidents}(\mathsf{date\_year} \to \mathsf{date\_month}, 12, 0)$
$\psi_2 = \mathsf{accidents}(\mathsf{local\_authority\_district} \to \mathsf{police\_force}, 2, 0)$
$\psi_3 = \mathsf{accidents}(\{\mathsf{date\_year}, \mathsf{date\_month}, \mathsf{data\_day}, \mathsf{police\_force}\} \to \mathsf{accident\_index}, 304, 0)$
$\psi_4 = \mathsf{vehicles}(\mathsf{accident\_index} \to \{\mathsf{accident\_index}, \mathsf{casualty\_reference}\}, 192, 0)$
$\psi_5 = \mathsf{casualty}(\{\mathsf{accident\_index}, \mathsf{casualty\_reference}\} \to \{\mathsf{accident\_index}, \mathsf{vehicle\_reference}\}, 9, 0)$
$\psi_6 = \mathsf{naptantraffic}(\mathsf{atcocode} \to \mathsf{accident\_index}, 3062, 0)$
$\psi_7 = \mathsf{accidents}(\mathsf{police\_force} \to \mathsf{localityname}, 12, 0)$
$\psi_8 = \mathsf{naptan}(\mathsf{localityname} \to \mathsf{atcocode}, 504, 0)$
$\psi_9 = \mathsf{accidents}(\mathsf{accident\_index} \to \mathsf{localityname}, 1, 0)$
$\psi_{10} = \mathsf{accidents}(\mathsf{date\_month} \to \mathsf{date\_day}, 31, 0)$
$\psi_{11} = \mathsf{naptan}(\mathsf{atcocode} \to \{\mathsf{latitude}, \mathsf{longitude}\}, 1, 0)$
$\psi_{12} = \mathsf{accidents}(\mathsf{casualty\_reference} \to \mathsf{age\_band\_of\_casualty}, 1)$

(2) TPCH .

$\psi_{13} = \mathsf{orders}(\{\mathsf{o\_custkey}, \mathsf{l\_orderdata}\} \to \mathsf{o\_orderkey}, 6, 0)$
$\psi_{14} = \mathsf{part}(\mathsf{p\_retailprice} \to \mathsf{p\_partkey}, 10, 0)$
$\psi_{15} = \mathsf{lineitem}(\{\mathsf{l\_shipdate}, \mathsf{l\_commitdate}\} \to \mathsf{l\_extendedprice}, 45, 0)$
$\psi_{16} = \mathsf{lineitem}(\mathsf{l\_shipdate} \to \mathsf{l\_quantity}, 50, 0)$
$\psi_{17} = \mathsf{custom}(\mathsf{c\_acctbal} \to \mathsf{c\_custkey}, 4, 0)$
$\psi_{18} = \mathsf{partsupp}(\mathsf{ps\_suppkey} \to \{\mathsf{ps\_partkey}, \mathsf{ps\_availqty}, \mathsf{ps\_supplycost}\}, 80, 0)$
$\psi_{19} = \mathsf{lineitem}(\mathsf{l\_receiptdate} \to \mathsf{l\_discount}, 11, 0)$
$\psi_{20} = \mathsf{lineitem}(\mathsf{l\_receiptdate} \to \mathsf{l\_discount}, 11, 0)$
$\psi_{21} = \mathsf{orders}(\mathsf{o\_orderstatus} \to \mathsf{o\_clerk}, 1000)$

(3) AIRCA .

$\psi_{22} = \mathsf{Market}(\mathsf{Year} \to \mathsf{Month}, 12, 0)$
$\psi_{23} = \mathsf{Market}(\mathsf{Month} \to \mathsf{Airline\_ID}, 256, 0)$
$\psi_{24} = \mathsf{OnTimePerformance}(\mathsf{OriginCityName} \to \mathsf{OriginAirportID}, 3, 0)$
$\psi_{25} = \mathsf{OnTimePerformance}(\mathsf{AirlineID} \to \mathsf{DepTime}, 1441, 0)$
$\psi_{26} = \mathsf{Segment}(\mathsf{Dest\_City\_Name} \to \mathsf{Dest\_Airport\_ID}, 10, 0)$
$\psi_{27} = \mathsf{OnTimePerformance}(\mathsf{OriginCityName} \to \mathsf{DestCityName}, 17, 0)$
$\psi_{28} = \mathsf{Market}(\mathsf{Origin\_City\_Name} \to \mathsf{Origin\_City\_Market\_ID}, 5, 0)$

(B) Access templates. We built access templates associated with these access constraints as described in Section 8.

Queries used in the experiments

Below are 30 TFACC queries, 30 AIRCA queries and 8 TPCH queries (we also used the 22 built-in TPCH queries).

q_1: **select** vehicles.casualty_reference
from vehicles, naptantraffic, naptan
where vehicles.accident_index = naptantraffic.accident_index **and**
naptan.latitude = 52.136436 **and** naptan.longitude = -0.460739 **and** naptan.atcocode = naptantraffic.atcocode

q_2: **select** vehicles.vehicle_reference
from vehicles, accidents
where vehicles.accident_index = accidents.accident_index **and**
accident.latitude = 50.106471 **and** accident.longitude = 0.490132

q_3: **select** Location_Easting_OSGR
from accidents
where naptan.latitude = 50.521306 **and** naptan.longitude = -0.417018 **and** data_year = 2000

q_4: **select** COUNT(vehicles.casualty_reference)
from vehicles, naptantraffic, naptan
where vehicles.accident_index = naptantraffic.accident_index **and**
naptan.latitude = 52.136436 **and** naptan.longitude = -0.460739 **and** naptan.atcocode = naptantraffic.atcocode

q_5: **select** accidents.data_year, COUNT(DISTINCT vehicletype)
from accidents, vehicles
where accidents.accident_index = vehicles.accident_index **and**
accidents.local_authority = 23 **and** vehicles.casualty_class = 3
group by year

q_6: **select** naptan.street
from accidents, vehicles, casualty, naptantraffic, naptan
where accidents.accident_index = vehicle.accident_index **and**
and accidents.sex_of_driver = casualty.sex_of_driver
and casualty_pedestrain_location = 0
and accidents.localtion_easting_osgr = naptantraffic.location
and naptantraffic.atcocode = naptan.atcocode
and naptan.localtyname = 'Penmachno'

q_7: **select** casualty.casualty_class, COUNT(DISTINCT casualty.casualty_reference)
from accidents, casualty
where accidents.date_year = 2001 **and** vehicles.date_month = 3
where accidents.accident_index = casualty.accident_index
and accidents.latitude **not in**
(**select** latitude
from naptan
where atcocode = 511)
and accidents.longitude **not in**
(**select** longitude
from naptantraffic
where localityname = 'Abergeeg')
group by casualty.casualty_class

q_8: **select** accidents.police_force
from accidents, naptantraffic, naptan
where accidents.data_year = 1995 **and** accidents.data_month = naptantraffic.creationtime_month **and**
accidents.data_year = naptantraffic.creationtime_year **and**
naptan.parentlocalityname = naptantraffic.localityname
naptan.localityname = 'Harmans Water'

q_9: **select** accidents.age_band_of_casualty, COUNT(DISTINCT casualty.casualty_reference)
from vehicles, accidents, casualty
where accidents.data_year = 2001 **and** accidents.police_force = 3 **and**
accidents.accident_index = vehicles.accident_index **and**
vehicles.age_of_vehicle $\leqslant$ 5 **and** vehicles.accident_index = casualty.accident_index
group by accidents.age_band_of_casualty

q_{10}: **select** vehicles.local_authority_district
from casualty, accidents
where casualty.sex_of_casualty = 1 **and** casualty.accident_index = accidents.accident_index **and**
casualty.pedestrain_location = 7 **and**
casualty.age_band_of_casualty = 3 **and**
casualty.casualty_severity = 1

q_{11}: **select** Airline_ID, SUM(Distance)
from Market
where Month = 1 **and** Dest_City_Market_ID = 30994
and Origin_City_Name = 'Albany'
group by Airline_ID

q_{12}: **select** O2.Dest_City_Name
from OnTimePerformance **as** O1, OnTimePerformance **as** O2
where O1.Carrier = 'UA'
and O1.OriginCityName = 'Kodiak'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier

q_{13}: **select** Segment.Carrier_Group
from OnTimePerformance **as** O1, OnTimePerformance **as** O2, Segment
where Segment.Airline_ID = O1.Airline_ID **and** O1.Carrier = 'UA'
and O1.OriginCityName = 'Kodiak'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier **and** O2.DestCityName = 'Buffalo'

q_{14}: **select** Segment.Air_Time
from OnTimePerformance **as** O1, OnTimePerformance **as** O2, Segment
where Segment.Dest_City_Name = O2.DestCityName **and** O1.Carrier = 'UA'
and O1.OriginCityName = 'Kodiak'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier

q_{15}: **select** Segment.Airline_ID, SUM(Segment.Payload)
from OnTimePerformance **as** O1, OnTimePerformance **as** O2, Segment, Market
where O1.Carrier = 'UA'
and O1.OriginCityName = 'Kodiak'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier
and Market.Dest_City_Market_ID = 30299
and Market.Dest_City_Name = O2.DestCityName
and O2.DestCityName = Segment.Dest_City_Name
group by Segment.Airline_ID

q_{16}: **select** O1.Carrier
from OnTimePerformance **as** O1, OnTimePerformance **as** O2
where O1.Carrier = 'UA'
and O1.OriginCityName = 'Kodiak'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier
and O1.DestCityName **not in**
(**select** DestCityName
from OnTimePerformance
where Carrier = 'UA' **and** OriginCityName = 'Kodiak')

q_{17}: **select** DestCityName
from OnTimePerformance
where (Carrier = 'UA' **or** Carrier = 'WN') **and** OriginAirportID = 10257

q_{18}: **select** O1.Carrier
from OnTimePerformance **as** O1, OnTimePerformance **as** O2
where (O1.Carrier = 'UA' **or** O1.Unique_Carrier_Entity = 1260)
and O1.OriginCityName = 'Kodiak'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier
and O1.DestCityName **not in**
(**select** DestCityName
from OnTimePerformance
where Carrier = 'UA' **and** OriginCityName = 'Kodiak')

q_{19}: **select** O1.OriginCityName
from OnTimePerformance **as** O1, OnTimePerformance **as** O2
where O1.Carrier = 'UA'
and O1.OriginCityName = 'Kodiak'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier

q_{20}: **select** COUNT(DISTINCT O1.Airline)
from OnTimePerformance **as** O1, OnTimePerformance **as** O2
where O1.Carrier = 'UA'
and O1.OriginCityName = 'Kodiak'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier

q_{21}: **select** vehicles.casualty_reference
from vehicles, naptantraffic, naptan
where vehicles.accident_index = naptantraffic.accident_index **and**
naptan.latitude = 52.701395 **and** naptan.longitude = -0.474912 **and** naptan.atcocode = naptantraffic.atcocode

q_{22}: **select** vehicles.vehicle_reference
from vehicles, accidents
where vehicles.accident_index = accidents.accident_index **and**
accident.latitude = 51.316724 **and** accident.longitude = -0.481369

q_{23}: **select** Location_Easting_OSGR
from accidents
where naptan.latitude = 51.497962 **and** naptan.longitude = -0.469322 **and** data_year = 1997

q_{24}: **select** COUNT(vehicles.casualty_reference)
from vehicles, naptantraffic, naptan
where vehicles.accident_index = naptantraffic.accident_index **and**
naptan.latitude = 53.014527 **and** naptan.longitude = -0.492851 **and** naptan.atcocode = naptantraffic.atcocode

q_{25}: **select** accidents.data_year, COUNT(DISTINCT vehicletype)
from accidents, vehicles
where accidents.accident_index = vehicles.accident_index **and**
accidents.local_authority = 18 **and** vehicles.casualty_class = 2
group by year

q_{26}: **select** naptan.street
from accidents, vehicles, casualty, naptantraffic, naptan
where accidents.accident_index = vehicle.accident_index **and**
and accidents.sex_of_driver = casualty.sex_of_driver
and casualty_pedestrain_location = 1
and accidents.localtion_easting_osgr = naptantraffic.location
and naptantraffic.atcocode = naptan.atcocode
and naptan.localtyname = 'Llandrillo-yn-Rhos'

q_{27}: **select** casualty.casualty_class, COUNT(DISTINCT casualty.casualty_reference)
from accidents, casualty
where accidents.date_year = 2002 **and** vehicles.date_month = 7
where accidents.accident_index = casualty.accident_index
and accidents.latitude **not in**
(**select** latitude
from naptan
where atcocode = 80)
and accidents.longitude **not in**
(**select** longitude
from naptantraffic
where localityname = 'Dolgarrog')
group by casualty.casualty_class

q_{28}: **select** accidents.police_force
from accidents, naptantraffic, naptan
where accidents.data_year = 1998 **and** accidents.data_month = naptantraffic.creationtime_month **and**
accidents.data_year = naptantraffic.creationtime_year **and**
naptan.parentlocalityname = naptantraffic.localityname
naptan.localityname = 'Caernarfon'

q_{29}: **select** accidents.age_band_of_casualty, COUNT(DISTINCT casualty.casualty_reference)
from vehicles, accidents, casualty
where accidents.data_year = 1995 **and** accidents.police_force = 2 **and**
accidents.accident_index = vehicles.accident_index **and**
vehicles.age_of_vehicle $\leqslant$ 6 **and** vehicles.accident_index = casualty.accident_index
group by accidents.age_band_of_casualty

q_{30}: **select** vehicles.local_authority_district
from casualty, accidents
where casualty.sex_of_casualty = 0 **and** casualty.accident_index = accidents.accident_index **and**
casualty.pedestrain_location = 0 **and**
casualty.age_band_of_casualty = 5 **and**
casualty.casualty_severity = 1

q_{31}: **select** Airline_ID, SUM(Distance)
from Market
where Month = 1 **and** Dest_City_Market_ID = 33214
and Origin_City_Name = 'Anchorage'
group by Airline_ID

q_{32}: **select** O2.Dest_City_Name
from OnTimePerformance **as** O1, OnTimePerformance **as** O2
where O1.Carrier = 'WN'
and O1.OriginCityName = 'Oakland'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier

q_{33}: **select** Segment.Carrier_Group
from OnTimePerformance **as** O1, OnTimePerformance **as** O2, Segment
where Segment.Airline_ID = O1.Airline_ID **and** O1.Carrier = 'WN'
and O1.OriginCityName = 'Oakland'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier **and** O2.DestCityName = 'Boston'

q_{34}: **select** Segment.Air_Time
from OnTimePerformance **as** O1, OnTimePerformance **as** O2, Segment
where Segment.Dest_City_Name = O2.DestCityName **and** O1.Carrier = 'WN'
and O1.OriginCityName = 'Oakland'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier

q_{35}: **select** Segment.Airline_ID, SUM(Segment.Payload)
from OnTimePerformance **as** O1, OnTimePerformance **as** O2, Segment, Market
where O1.Carrier = 'WN'
and O1.OriginCityName = 'Oakland'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier
and Market.Dest_City_Market_ID = 30299
and Market.Dest_City_Name = O2.DestCityName
and O2.DestCityName = Segment.Dest_City_Name
group by Segment.Airline_ID

q_{36}: **select** O1.Carrier
from OnTimePerformance **as** O1, OnTimePerformance **as** O2
where O1.Carrier = 'WN'
and O1.OriginCityName = 'Oakland'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier
and O1.DestCityName **not in**
(**select** DestCityName
from OnTimePerformance
where Carrier = 'WN') **and** OriginCityName = 'Oakland')

q_{37}: **select** DestCityName
from OnTimePerformance
where (Carrier = 'WN') **or** Carrier = 'CO') **and** OriginAirportID = 16440

q_{38}: **select** O1.Carrier
from OnTimePerformance **as** O1, OnTimePerformance **as** O2
where (O1.Carrier = 'WN' **or** O1.Unique_Carrier_Entity = 4350)
and O1.OriginCityName = 'Oakland'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier
and O1.DestCityName **not in**
(**select** DestCityName
from OnTimePerformance
where Carrier = 'WN' **and** OriginCityName = 'Oakland')

q_{39}: **select** O1.OriginCityName
from OnTimePerformance **as** O1, OnTimePerformance **as** O2
where O1.Carrier = 'WN'
and O1.OriginCityName = 'Oakland'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier

q_{40}: **select** COUNT(DISTINCT O1.Airline)
from OnTimePerformance **as** O1, OnTimePerformance **as** O2
where O1.Carrier = 'WN'
and O1.OriginCityName = 'Oakland'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier

q_{41}: **select** vehicles.casualty_reference
from vehicles, naptantraffic, naptan
where vehicles.accident_index = naptantraffic.accident_index **and**
naptan.latitude = 50.216318 **and** naptan.longitude = -0.511802 **and** naptan.atcocode = naptantraffic.atcocode

q_{42}: **select** vehicles.vehicle_reference
from vehicles, accidents
where vehicles.accident_index = accidents.accident_index **and**
accident.latitude = 51.446812 **and** accident.longitude = -0.471184

q_{43}: **select** Location_Northing_OSGR
from accidents
where naptan.latitude = 52.141803 **and** naptan.longitude = -0.475002 **and** data_year = 2003

q_{44}: **select** COUNT(vehicles.casualty_reference)
from vehicles, naptantraffic, naptan
where vehicles.accident_index = naptantraffic.accident_index **and**
naptan.latitude = 54.100156 **and** naptan.longitude = -0.419001 **and** naptan.atcocode = naptantraffic.atcocode

q_{45}: **select** accidents.data_year, COUNT(DISTINCT vehicletype)
from accidents, vehicles
where accidents.accident_index = vehicles.accident_index **and**
accidents.local_authority = 31 **and** vehicles.casualty_class = 2
group by year

q_{46}: **select** naptan.street
from accidents, vehicles, casualty, naptantraffic, naptan
where accidents.accident_index = vehicle.accident_index **and**
and accidents.sex_of_driver = casualty.sex_of_driver
and casualty_pedestrain_location = 3
and accidents.localtion_easting_osgr = naptantraffic.location
and naptantraffic.atcocode = naptan.atcocode
and naptan.localtyname = 'Merthyr Tydfil'

q_{47}: **select** casualty.casualty_class, COUNT(DISTINCT casualty.casualty_reference)
from accidents, casualty
where accidents.date_year = 2002 **and** vehicles.date_month = 5
where accidents.accident_index = casualty.accident_index
and accidents.latitude **not in**
(**select** latitude
from naptan
where atcocode = 609)
and accidents.longitude **not in**
(**select** longitude
from naptantraffic
where localityname = 'Glascoed')
group by casualty.casualty_class

q_{48}: **select** accidents.police_force
from accidents, naptantraffic, naptan
where accidents.data_year = 2004 **and** accidents.data_month = naptantraffic.creationtime_month **and**
accidents.data_year = naptantraffic.creationtime_year **and**
naptan.parentlocalityname = naptantraffic.localityname
naptan.localityname = 'Owlsmoor'

q_{49}: **select** accidents.age_band_of_casualty, COUNT(DISTINCT casualty.casualty_reference)
from vehicles, accidents, casualty
where accidents.data_year = 1998 **and** accidents.police_force = 2 **and**
accidents.accident_index = vehicles.accident_index **and**
vehicles.age_of_vehicle ⩽ 3 **and** vehicles.accident_index = casualty.accident_index
group by accidents.age_band_of_casualty

q_{50}: **select** vehicles.local_authority_district
from casualty, accidents
where casualty.sex_of_casualty = 0 **and** casualty.accident_index = accidents.accident_index **and**
casualty.pedestrain_location = 2 **and**
casualty.age_band_of_casualty = 2 **and**
casualty.casualty_severity = 3

q_{51}: **select** Airline_ID, SUM(Distance)
from Market
where Month = 1 **and** Dest_City_Market_ID = 30466
and Origin_City_Name = 'Casper'
group by Airline_ID

q_{52}: **select** O2.Dest_City_Name
from OnTimePerformance **as** O1, OnTimePerformance **as** O2
where O1.Carrier = 'CO'
and O1.OriginCityName = 'Philadelphia'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier

q_{53}: **select** Segment.Carrier_Group
from OnTimePerformance **as** O1, OnTimePerformance **as** O2, Segment
where Segment.Airline_ID = O1.Airline_ID **and** O1.Carrier = 'CO'
and O1.OriginCityName = 'Philadelphia'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier **and** O2.DestCityName = 'Nashville'

q_{54}: **select** Segment.Air_Time
from OnTimePerformance **as** O1, OnTimePerformance **as** O2, Segment
where Segment.Dest_City_Name = O2.DestCityName **and** O1.Carrier = 'CO'
and O1.OriginCityName = 'Philadelphia'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier

q_{55}: **select** Segment.Airline_ID, SUM(Segment.Payload)
from OnTimePerformance **as** O1, OnTimePerformance **as** O2, Segment, Market
where O1.Carrier = 'CO'
and O1.OriginCityName = 'Philadelphia'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier
and Market.Dest_City_Market_ID = 32457
and Market.Dest_City_Name = O2.DestCityName
and O2.DestCityName = Segment.Dest_City_Name
group by Segment.Airline_ID

q_{56}: **select** O1.Carrier
from OnTimePerformance **as** O1, OnTimePerformance **as** O2
where O1.Carrier = 'CO'
and O1.OriginCityName = 'Philadelphia'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier
and O1.DestCityName **not in**
(**select** DestCityName
from OnTimePerformance
where Carrier = 'CO' **and** OriginCityName = 'Philadelphia')

q_{57}: **select** DestCityName
from OnTimePerformance
where (Carrier = 'CO' **or** Carrier = 'EA') **and** OriginAirportID = 10529

q_{58}: **select** O1.Carrier
from OnTimePerformance **as** O1, OnTimePerformance **as** O2

where (O1.Carrier = 'CO' **or** O1.Unique_Carrier_Entity = 10220)
and O1.OriginCityName = 'Philadelphia'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier
and O1.DestCityName **not in**
(**select** DestCityName
from OnTimePerformance
where Carrier = 'CO' **and** OriginCityName = 'Philadelphia')

q_{59}: **select** O1.OriginCityName
from OnTimePerformance **as** O1, OnTimePerformance **as** O2
where O1.Carrier = 'CO'
and O1.OriginCityName = 'Philadelphia'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier

q_{60}: **select** COUNT(DISTINCT O1.Airline)
from OnTimePerformance **as** O1, OnTimePerformance **as** O2
where O1.Carrier = 'CO'
and O1.OriginCityName = 'Philadelphia'
and O1.DestCityName = O2.OriginCityName **and** O2.Carrier = O1.Carrier

q_{61}: **select** p2.p_brand, COUNT(DISTINCT p2.p_container)
from part **as** p1, part **as** p2
where p1.p_retailprice = 905.00 **and**
p1.p_mfgr = p2.p_mfgr **and**
p2.p_type = 'ECONOMY ANODIZED COPPER' **and** p2.p_size = 3;
goup by p2.p_brand

q_{62}: **select** p2.p_mfgr, SUM(p2.p_retailprice)
from part **as** p1, part **as** p2
where p1.p_mfgr = 'Manufacturer#1' **and** p1.p_brand = p2.p_brand;
group by p2.p_mfgr

q_{63}: **select** l_orderkey, l_extendedprice, l_discount, o_orderdate, o_shippriority
from customer, orders, lineitem
where c_mktsegment = 'BUILDING' **and** c_custkey = o_custkey
and l_orderkey = o_orderkey **and** o_orderdate = '1995-03-15' **and** l_shipdate = '1995-03-25'

q_{64}: **select** SUM(l_extendedprice)
from customer, orders, lineitem
where c_acctball $\geqslant$ 200000 **and** c_custkey = o.custkey
o.orderdate_year = 1995 **and** o.orderdate_month = 6 **and** o_orderkey = l_orderkey

q_{65}: **select** l_extendedprice
from lineitem, part
where p_partkey = l_partkey **and**
(p_brand = 'Brand#12' **or** p_brand = 'Brand#37') **and** p_container **in**
(**select** DISTINCT p2.p_container
from part **as** p1, part **as** p2
where p1.p_mfgr = 'Manufacturer#3' **and** p1.p_brand = p2.p_brand **and**
p1.p_type = 'ECONOMY ANODIZED COPPER' **and** p2.p_size = 3)
and p_size $\geqslant$ 1 **and** p_size $\leqslant$ 4

q_{66}: **select** l_extendedprice, l_discount
from lineitem, part
where p_partkey = l_partkey **and**
(p_brand = 'Brand#12' **or** p_brand = 'Brand#37') **and** p_container **not in**
(**select** p_size **from** part **where** p_mfgr = 'Manufacturer#3' **and** p_type = 'SMALL PLATED BRASS')
and p_size $\geqslant$ 1 **and** p_size $\leqslant$ 4

q_{67}: **select** l2.tax
from lineitem **as** l1, lineitem **as** l2
where l1.receiptdate = date '1995-09-01' **and** l1.commitdate = l2.shipdate

q_{68}: **select** o_totalprice
from part, partsupp, lineitem, orders
where p_mfgr = 'Manufacturer#3' **and** p_type = 'ECONOMY ANODIZED COPPER' **and**
p_size = 3 **and** p_partkey = ps_partkey **and** ps_suppkey = l_suppkey **and** l_shipdate = o_orderdate