Data Driven Approximation with Bounded Resources

Yang Cao[1,2] Wenfei Fan[1,2]
[1]University of Edinburgh [2]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 [36]. 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 [24]. 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) $\mathsf{accuracy}(Q(D_Q), Q, D) \geqslant \eta$.

Proceedings of the VLDB Endowment, Vol. 10, No. 9

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, 29], (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 [21]:

```
select  h.address, h.price
from    poi as h, friend as f, person as p
where   f.pid = p0 and f.fid = p.pid and p.city = h.city
        and h.type = "hotel" and h.price ⩽ 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 [25]. 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} $\rightarrow$ {price, address}, 1, (e_p^1, e_a^1)),
  ...
- ψ_m: poi({type, city} $\rightarrow$ {price, address}, 2^m, (e_p^m, e_a^m)),
- φ_1: friend(pid $\rightarrow$ fid, 5000, 0),
- φ_2: person(pid $\rightarrow$ 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 [25], 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 [11,22] 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, 29] is allowed. As opposed to prior accuracy metrics [17, 26, 27, 33], 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 dataset D, there exists an access schema $\mathcal{A}$ such that D conforms to $\mathcal{A}$, and for all resource ratios $\alpha \in (0, 1]$ and queries Q 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 α is, the higher η is.

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).

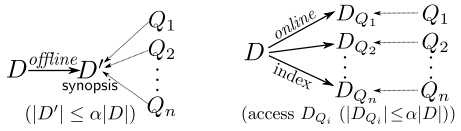
(5) Empirical study (Section 8). Using real-life and synthetic data, we experimentally verify the effectiveness of BEAS. We find the following. (a) BEAS computes approximate answers with accuracy $\eta \geqslant 0.85$ for SPC, and $\eta \geqslant 0.82$ for RA, aggregate or not, when $\alpha \geqslant 5.5 \times 10^{-4}$. 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 [27] and BlinkDB [8] under RC measure and MAC measure [27]. 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, 22, 23]. Under a set $\mathcal{A}$ of access constraints introduced in [23] a query Q is *boundedly evaluable* [22] 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 $\mathcal{A}$ [23], under $\mathcal{A}$ [22], and with views [12]. The problem is shown undecidable for Q in RA and EXPSPACE -hard for Q in SPC [22]. 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.



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

Figure 1: Data reduction schemes

(1) Access schema. The access constraints of [11–13, 22, 23] 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 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, 22, 23].

Approximation. Approximate query answering is typically based on either (a) synopsis [7, 14, 16, 27, 28], or (b) dynamic sampling (*e.g.,* [8, 9, 19, 30]). 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, 19]. 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, 19, 30]. 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, 27, 28]. While [8, 19, 30] use dynamic sampling, [19, 30] do not support bounded data access; [8] employs stratified sampling; given an error bar on the value distribution of an aggregate attribute, [19] 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, 32, 34]. 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.

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 schemas. 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 [11,23] 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, 23].

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. *A plan* ξ *under* $\mathcal{A}$ is a sequence of RA operators and an operator $\mathsf{fetch}(X \in T, R, Y, \psi)$, where ψ is an access template $R(X \to Y, N, \bar{d}_Y)$ in $\mathcal{A}$ and T is an intermediate relation returned by some operator prior to the

fetch in ξ. The fetch retrieves a set $W_{\bar{a}} = \tilde{D}_Y^N(X = \bar{a})$ for each $\bar{a} \in T$ from D; it returns $\bigcup_{\bar{a}\in T}\{(\bar{a}, \bar{b}) \mid \bar{b} \in W_{\bar{a}}\}$.

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, 22, 23] 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:

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

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* $d(t, t') = \mathsf{max}_{A\in\mathsf{R}}|\mathsf{dis}_A(t[A], t'[A])|$ for tuples t and t' of relation R, the worst 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

$$\mathsf{min}_{s\in S} d(s, t).$$

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 $\mathsf{min}_{t\in Q(D)} d(s, t)$, 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, 29].

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

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

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 [33]; (b) distance-based, *e.g.,* Hausdorff distance [26] and MAC [27], using distance functions to measure either the gap between approximate and exact answers [27], 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 Aggregate Queries

We now extend the distance functions to aggregate queries. We consider $\mathsf{RA}_{\mathsf{aggr}}$ [20], 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 $d_{\mathsf{agg}}(s, t) = \mathsf{max}\{\mathsf{max}_{A \in X} |\mathsf{dis}_A(s[A], t[A])| + f_{\mathsf{agg}}(t[V], s[V])\}$.

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 $\{t_1 = (\text{NYC}, 100), t_2 = (\text{Chicago}, 140)\}$, and S is $\{s_1 = (\text{NYC}, 80), s_2 = (\text{Chicago}, 150)\}$. 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 $\delta_{\mathsf{cov}}(Q_1', D, t_1) = \mathsf{max}\{0, |100 - 80|\} = 20$, $\delta_{\mathsf{cov}}(Q_1', D, t_2) = \{0, |140 - 150|\} = 10$, measuring the difference in count. □

4. RESOURCE BOUNDED EVALUATION

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, resource ratio $\alpha \in (0, 1]$ and dataset $D \models \mathcal{A}$, it generates (a) an α-bounded query plan ξ_α for Q 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.

Implementation. For each relation $R \in \mathcal{R}$ and instance D_R of R, we can build indices on D_R for templates $\psi_1^R, \ldots, \psi_{M_R}^R$ in three steps as follows. (1) Build a K-D tree [35] T_D of D_R by treating tuples as m_R-dimensional points *w.r.t.* distance functions, where $m_R = |\mathsf{attr}(R)|$. (2) For each $k \in [1, M_R]$, the index for ψ_k^R is a table $T_k^R(I, \mathsf{attr}(R))$

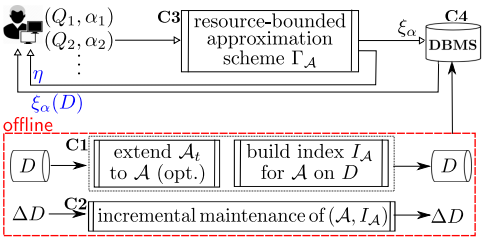


Figure 2: Workflow of BEAS

such that $\pi_I(T_k^R) = \{k\}$ and $\pi_{\mathsf{attr}(R)}(T_k^R)$ consists of tuples at the k-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. (3) Store the indices for ψ_1^R, $\ldots$, $\psi_{M_R}^R$ in a single table $T_R(I, \mathsf{attr}(R))$ by taking the union of T_k's, and build an hash index on I over T_R.

The indices can be readily extended to incorporate access constraints and access templates that extend $\mathcal{A}_t$.

There are other approaches to constructing the indices of $\mathcal{A}_t$. We adopt K-D trees for two reasons. (1) K-D trees are easy to implement and maintain [35]. (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.

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 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; **C3**). It executes ξ_α by the DBMS, accessing at most $\alpha|D|$ tuples (**C4**). It returns $(\xi_\alpha(D), \eta)$. Underlying the approximation scheme are the algorithms for **C3**, to be given in Sections 5, 6 and 7. The plan ξ_α is executed directly by the underlying DBMS (**C4**).

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 2: *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.* □

Algorithm $\mathsf{BEAS}_{\mathsf{SPC}}$

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 $L_{\psi_k^\delta \to \psi_{k+1}^\delta}(\xi_\alpha) \geqslant L_{\psi_k^{\delta'} \to \psi_{k+1}^{\delta'}}(\xi_\alpha)$ $(\forall \delta')$;
4. $\quad$ replace template ψ_k^δ with ψ_{k+1}^δ for fetch δ in ξ_α;
5. **return** (ξ_α, η);

Figure 3: Algorithm $\mathsf{BEAS}_{\mathsf{SPC}}$

Here Σ_3^p is the complexity class at the third level of the polynomial hierarchy [37], beyond NP unless P = NP.

We show that the Σ_3^p-hardness even when $\eta = 1$ and $\mathcal{A}$ is $\mathcal{A}_t$ of Section 4.1, by reduction from the $\exists^*\forall^*\exists^*$CNF problem, which is Σ_3^p-complete [37]. 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 bound η.

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

Approximation scheme $\mathsf{BEAS}_{\mathsf{SPC}}$. The scheme is denoted by $\mathsf{BEAS}_{\mathsf{SPC}}$ and shown in Fig. 3. Given an SPC query Q, an access schema $\mathcal{A}$ that subsumes $\mathcal{A}_t$, and budget $B = \alpha|D|$, $\mathsf{BEAS}_{\mathsf{SPC}}$ 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.

$\mathsf{BEAS}_{\mathsf{SPC}}$ 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".

Lemma 3: *Under access schema $\mathcal{A} \supseteq \mathcal{A}_t$, every* SPC *query Q has a canonical bounded plan.* □

$\mathsf{BEAS}_{\mathsf{SPC}}$ 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, $\mathsf{BEAS}_{\mathsf{SPC}}$ builds a *chasing sequence* ℓ for Q under $\mathcal{A}$ via $\mathsf{chase}(Q, \mathcal{A}, B)$ (line 1, Fig. 3), 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

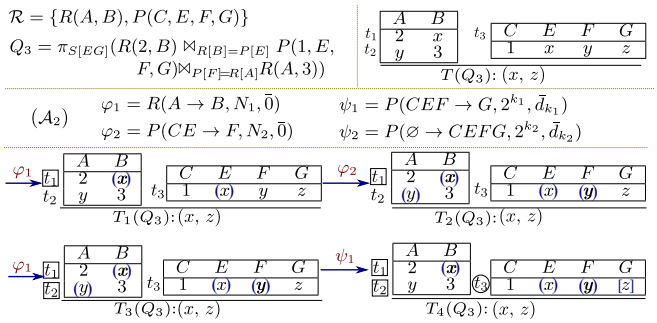


Figure 4: A chasing sequence for Q_3 under $\mathcal{A}_2$

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. 4, in which constants and variables represent 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. 4); 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.

(b) Tuple marking. If all variables in t are exactly covered by γ_i, then tuple t is also marked *exactly covered*; otherwise, if all its variables are covered, then t is *approximately covered*.

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.

An approximately covered variable (tuple) can possibly be "upgraded" to exactly covered but not the other way around.

One can verify the following.

Lemma 4: *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.* □

Example 4: Consider query Q_3 and access schema $\mathcal{A}_2$ given in Fig. 4. A chasing sequence of 5 steps for Q_3 under $\mathcal{A}_2$ is depicted at the bottom of Fig. 4. 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. 4, 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 5: From the chasing sequence of Fig. 4, 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: for each $\sigma_{A=c}$, if A is fetched via $R(X \to Y, 2^k, \bar{d}_k)$ in $\mathcal{A}$, where $A \in Y$ and c is a constant, then replace $\sigma_{A=c}$ with a relaxed condition $\sigma_{|\mathsf{dis}_A(A,c)| \leqslant \bar{d}_k[A]}$; similarly for $\sigma_{A=B}$.

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 5, 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. $\mathsf{BEAS}_{\mathsf{SPC}}$ 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. 3).

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). 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. When Q is $\sigma_{R[A]=c}(Q')$, (a) $d_{\mathsf{cov}}(Q) = d_{\mathsf{cov}}(Q')$, and (b) $d_{\mathsf{rel}}(Q) = \mathsf{max}\{d_{\mathsf{rel}}(Q'), \bar{d}_k[A]\}$ if $R[A]$ is fetched using access template $R(X \to Y, 2^k, \bar{d}_k)$, and $d_{\mathsf{rel}}(Q) = d_{\mathsf{rel}}(Q')$ otherwise.

Intuitively, the worst coverage distance of Q is the same as that of Q'. The worst relevance distance in D for an approximate answer s and Q is the worst between the distance for s and Q' in D and resolution $\bar{d}_k[A]$ if s is approximately covered using a template with $\bar{d}_k$. If s is exactly covered, *i.e.,* by an access constraint, no extra distance is introduced by fetch.

The cases of R, $\sigma_{R[A]\leqslant c}(Q')$, $\pi_Y(Q')$, $\sigma_{R[A]=R'[B]}(Q')$, $\sigma_{R[A]\leqslant R'[B]}(Q')$ and $Q_1 \times Q_2$ are similar.

Example 6: Given Q_1 and $\mathcal{A}_0$ from Example 1, $\mathsf{BEAS}_{\mathsf{SPC}}$ returns exactly the plan in Example 1 for Q_1 under $\mathcal{A}_0 \cup \mathcal{A}_t$, and $\eta = 1/(1 + \mathsf{max}\{e_p^{k_\alpha}, e_a^{k_\alpha}\})$. 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 $\mathsf{max}\{e_p^{k_\alpha}, e_a^{k_\alpha}\}$ (resp. $e_p^{k_\alpha}$), which gives the η. □

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

Theorem 5: *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) $\eta \geqslant 1/(1 + \mathsf{max}_{\varphi \in \mathcal{A}} \bar{d}^m_{(\psi, k^*)})$*, where* $k^* = \lfloor \log_2 \frac{\alpha|D|}{\|Q\|} \rfloor - 1$*;*

(3) and 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 $\psi_k = R(X \to Y, 2^k, \bar{d}_k)$ in $\mathcal{A}_0$, we denote $\mathsf{max}_{A \in Y} \bar{d}_k[A]$ as $\bar{d}^m_{(\psi,k)}$; (2) for a query Q and $\|Q\|$ is the number of relations in Q; and (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 $1/(1 + \mathsf{max}_{\varphi \in \mathcal{A}} \bar{d}^m_{(\psi, k^*)}) > 0$. This verifies Theorem 1 for SPC. (2) Both η and the lower bound $1/(1 + \mathsf{max}_{\varphi \in \mathcal{A}} \bar{d}^m_{(\psi, k^*)})$ for η depend on Q since $\mathsf{BEAS}_{\mathsf{SPC}}$ advocates dynamic data reduction guided by the structure of Q 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 $1/(1 + \mathsf{max}_{\varphi \in \mathcal{A}} \bar{d}^m_{(\psi, k^*)})$. 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 $\mathsf{max}_{\varphi \in \mathcal{A}} \bar{d}^m_{(\psi, k^*)}$ estimated above by using Q and $\mathcal{A}$ alone via $\mathsf{BEAS}_{\mathsf{SPC}}$. (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.

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. 5. 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 3 holds for RA. Hence $\mathsf{BEAS}_{\mathsf{RA}}$ also generates canonical bounded plans for RA queries.

Generating initial fetching plan ξ_F. Similar to $\mathsf{BEAS}_{\mathsf{SPC}}$, $\mathsf{BEAS}_{\mathsf{RA}}$ 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'. $\mathsf{BEAS}_{\mathsf{RA}}$ generates fetching plans for all max SPC sub-queries of Q via $\mathsf{BEAS}_{\mathsf{SPC}}$, 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. $\mathsf{BEAS}_{\mathsf{RA}}$ then chooses access templates with higher resolution for ξ_F via chAT (step (2) of $\mathsf{BEAS}_{\mathsf{SPC}}$), 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, $\mathsf{BEAS}_{\mathsf{RA}}$ 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 $\mathsf{BEAS}_{\mathsf{SPC}}$.

(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 in-

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 *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;
   */*let $\widehat{d_r^\alpha}$ and $\widehat{d_c^\alpha}$ be the values of d_{rel} and d_{cov} in $L(\widehat{\xi_\alpha})$ */*
6. $S := \xi_\alpha(D)$; $\widehat{S} := \widehat{\xi_\alpha}(D)$; */* executing ξ_α and $\widehat{\xi_\alpha}$ */*
7. $d' := \mathsf{max}_{t \in S'} \mathsf{min}_{s \in S} \delta_{\mathsf{cov}}(\widehat{Q}, t, s)$; $\eta' := \frac{1}{1+\mathsf{max}(d_r^\alpha, d' + \widehat{d_c^\alpha})}$;
8. **return** (ξ_α, η');

Figure 5: Algorithm $\mathsf{BEAS_{RA}}$

volved, such that for any $D \models \mathcal{A}$, if $t \in Q_2(D)$ then t must be in $\pi_{R_{Q_1}} \sigma_C(\mathsf{E}(Q_1) \times \mathsf{E}(\widehat{Q}_2))(D)$.

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, $\mathsf{BEAS_{RA}}$ first generates an α-bounded plan $\widehat{\xi_\alpha}$ for the maximal induced query $\widehat{Q}$ of Q using $\mathsf{BEAS_{SPC}}$ (lines 4-5). It then replaces coverage distance bound d_c^α in $L(\xi_F)$ with $\widehat{d_c^\alpha} + d'$, where $d' = \mathsf{max}_{t \in \widehat{s}} \mathsf{min}_{s \in s} \delta_{\mathsf{cov}}(\widehat{Q}, t, s)$, the "coverage distance" between tuples in S and S' for $\widehat{Q}$, in which $S = \xi_\alpha(D)$ and $\widehat{S} = \widehat{\xi_\alpha}(D)$, yielding a correct accuracy bound $\eta' = \frac{1}{1+\mathsf{max}(d_r^\alpha, d' + \widehat{d_c^\alpha})}$ (lines 6-7). Here $\widehat{d_c^\alpha}$ (resp. d_r^α) is the coverage bound in $L(\widehat{\xi_F})$ (resp. $L(\xi_F)$) chosen in line 2 (resp. line 5).

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 7: Consider RA query $Q_4 = \pi_B(R(1,B) - W(1,B))$. 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 is $|\mathsf{dis}_B(R[B], W[B])| \leqslant \bar{d}_{k_W}[B]$. 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.

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 in $\mathsf{BEAS_{RA}}$ is $\bar{d}_{k_R}[B]$. Assume that t_2 is removed by $\mathsf{E}(Q_4)$ from the answers. Then d_c^α is not an upper bound on the coverage distance $\delta_{\mathsf{cov}}(Q_4, t, s)$.

To rectify this, $\mathsf{BEAS_{RA}}$ computes upper bound $\widehat{d_c^\alpha}$ on coverage of $\widehat{Q}_4$ and the "coverage distance" d' for using t_1 only to cover $\{t_1, t_2\}$. It takes $\widehat{d_c^\alpha} + d'$ as an upper bound for $\delta_{\mathsf{cov}}(Q_4, t, s)$; this makes a lower bound on the distance. □

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

Theorem 6: *For any* RA Q*, access schema $\mathcal{A} \supseteq \mathcal{A}_t$, resource ratio α and dataset $D \models \mathcal{A}$,* $\mathsf{BEAS_{RA}}$ *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) $\mathsf{F_{rel}}(\xi_\alpha(D), Q, D) \geqslant \eta$ *and* $\mathsf{F_{cov}}(\xi_\alpha(D), Q, D) \geqslant \eta$*;*

(2) $\mathsf{F_{rel}}(\xi_\alpha(D), Q, D) \geqslant 1/(1 + \mathsf{max}_{\psi \in \mathcal{A}} \bar{d}^m_{(\psi, k^*)})$*, 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 5. □

$\mathsf{BEAS_{RA}}$ 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) Both η and the lower bound depends on Q, by employing dynamic reduction. Moreover, similar to $\mathsf{BEAS_{SPC}}$, 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.

7. APPROXIMATING AGGREGATION

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

Handling max and min. We start with $\mathsf{RA_{aggr}}$ queries Q defined with max and min. One can verify that Lemma 3 also holds on Q. $\mathsf{BEAS_{agg}}$ generates canonical bounded plans $\xi_\alpha = (\xi_F, \xi_E)$ for Q. It works in the same way as $\mathsf{BEAS_{RA}}$ except the following extensions. Let $Q = \mathsf{gpBy}(Q', X, \mathsf{agg}(V))$.

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 $\mathsf{BEAS_{RA}}$.

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

Corollary 7: *For ξ_α and η generated by* $\mathsf{BEAS_{agg}}$*, the lower bounds and properties on ξ_α and η specified in Theorem 6 remain the same for any* $\mathsf{RA_{aggr}}$ *query with* min *and* max. □

For $\mathsf{RA_{aggr}}$ 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, $\mathsf{BEAS_{agg}}$ can be extended.

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 $i \in [0, \lceil \log_2 \max_{\bar{a} \in D_{XY}} |D_Z(XY = \bar{a})| \rceil]$, 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.

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 derived 30 $\mathsf{RA}_{\mathsf{aggr}}$ queries from its built-in SQL queries.

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 [27].

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 [27], 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.

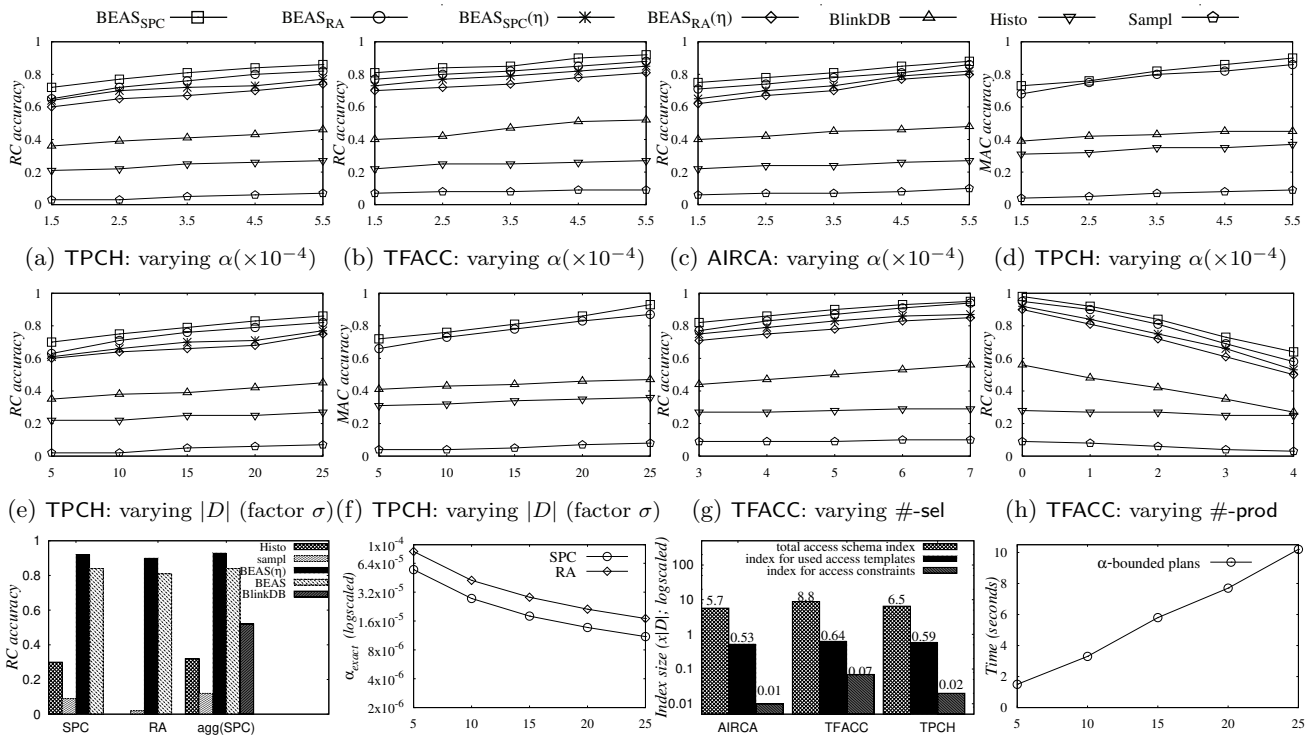
(A) RC-measure. We find the following. As reported in Figures 6(a), 6(b), 6(c), (a) the approximate answers computed by BEAS are accurate. $\mathsf{BEAS}_{\mathsf{SPC}}$ and $\mathsf{BEAS}_{\mathsf{RA}}$ are consistently above 0.72; $\mathsf{BEAS}_{\mathsf{SPC}}$ is above 0.85 when $\alpha \geqslant 3.5 \times 10^{-4}$, 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. 6(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. 6(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. 6(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



(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 6: Accuracy and scalability of BEAS, the resource-bounded approximation framework

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. 6(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 6(g), 6(h) and 6(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 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.

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 6(a), 6(b), 6(c) (varying α), Figure 6(e) (varying $|D|$) and Figures 6(g), 6(h) and 6(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. 6(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. 6(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 [31]. (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. 6(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.

Acknowledgments. Cao and Fan are supported in part by ERC 652976; NSFC 61421003; 973 Program 2014CB340302; EPSRC EP/M025268/1; Shenzhen Peacock Program 1105100030834361; Guangdong Innovative Research Team Program 2011D005; the Foundation for Innovative Research Groups of NSFC; Beijing Advanced Innovation Center for Big Data and Brain Computing (BDBC), Beihang University; and two Innovative Research Grants from Huawei Technologies. Cao is also supported in part by NSFC 61602023.

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] B. Ding, S. Huang, S. Chaudhuri, K. Chakrabarto, and C. Wang. Sample + seek: Approximating aggregates with distribution precision guarantee. In *SIGMOD*, 2016.
[20] R. Elmasri. *Fundamentals of database systems*. Pearson Education India, 2008.
[21] Facebook. Introducing Graph Search. *https://en-gb.facebook.com/about/graphsearch*, 2013.
[22] W. Fan, F. Geerts, Y. Cao, and T. Deng. Querying big data by accessing small data. In *PODS*, 2015.
[23] W. Fan, F. Geerts, and L. Libkin. On scale independence for querying big data. In *PODS*, 2014.
[24] W. Fan, X. Wang, and Y. Wu. Distributed graph simulation: Impossibility and possibility. *PVLDB*, 7(12), 2014.
[25] I. Grujic, S. Bogdanovic-Dinic, and L. Stoimenov. Collecting and analyzing data from e-government Facebook pages. In *ICT Innovations*, 2014.
[26] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the Hausdorff distance. In *TPAMI*, 1993.
[27] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *VLDB*, 1999.
[28] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, 2009.
[29] A. Kadlag, A. V. Wanjari, J. Freire, and J. R. Haritsa. Supporting exploratory queries in databases. In *DASFAA*, 2004.
[30] 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.
[31] C. Levine. TPC benchmarks. *http://research.microsoft.com /en-us/um/people/gray/WICS_99_TP/22_Levine.ppt*.
[32] C. Li. Computing complete answers to queries in the presence of limited access patterns. *VLDB J.*, 12(3), 2003.
[33] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. Xu. Regret-minimizing representative databases. In *VLDB*, 2010.
[34] A. Nash and B. Ludäscher. Processing first-order queries under limited access patterns. In *PODS*, 2004.
[35] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.
[36] Stack Overflow. SQL: Inner joining two massive tables. *http://stackoverflow.com/questions/1750001/sql-inner-joining-two-massive-tables*.
[37] L. J. Stockmeyer.The polynomial-time hierarchy. *TCS*, 3(1), 1976.