Querying Big Data with Bounded Data Access

Yang Cao

Doctor of Philosophy

Laboratory for Foundations of Computer Science

School of Informatics

University of Edinburgh

2017

Abstract

Query answering over big data is cost-prohibitive. A linear scan of a dataset D may take days with a solid state device if D is of PB size and years if D is of EB size. In other words, polynomial-time (PTIME) algorithms for query evaluation are already not feasible on big data. To tackle this, we propose querying big data with bounded data access, such that the cost of query evaluation is independent of the scale of D.

First of all, we propose a class of *boundedly evaluable* queries. A query Q is boundedly evaluable under a set $\mathcal{A}$ of access constraints if for *any* dataset D that satisfies constraints in $\mathcal{A}$, there exists a subset $D_Q \subseteq D$ such that (a) $Q(D_Q) = Q(D)$, and (b) the time for identifying D_Q from D, and hence the size $|D_Q|$ of D_Q, are *independent of* $|D|$. That is, we can compute $Q(D)$ by accessing a bounded amount of data no matter how big D grows. We study the problem of deciding whether a query is boundedly evaluable under $\mathcal{A}$. It is known that the problem is undecidable for FO without access constraints. We show that, in the presence of access constraints, it is decidable in 2EXPSPACE for positive fragments of FO queries, but is already EXPSPACE-hard even for CQ.

To handle the undecidability and high complexity of the analysis, we develop effective syntax for boundedly evaluable queries under $\mathcal{A}$, referred to as queries *covered* by $\mathcal{A}$, such that, (a) any boundedly evaluable query under $\mathcal{A}$ is equivalent to a query covered by $\mathcal{A}$, (b) each covered query is boundedly evaluable, and (c) it is efficient to decide whether Q is covered by $\mathcal{A}$. On top of DBMS, we develop practical algorithms for checking whether queries are covered by $\mathcal{A}$, and generating bounded plans if so.

For queries that are not boundedly evaluable, we extend bounded evaluability to resource-bounded approximation and bounded query rewriting using views. (1) Resource-bounded approximation is parameterized with a resource ratio $\alpha \in (0,1]$, such that for any query Q and dataset D, it computes approximate answers with an accuracy bound η by accessing at most $\alpha|D|$ tuples. It is based on extended access constraints and a new accuracy measure. (2) Bounded query rewriting tackles the problem by incorporating bounded evaluability with views, such that the queries can be exactly answered by accessing cached views and a bounded amount of data in D. We study the problem of deciding whether a query has a bounded rewriting, establish its complexity bounds, and develop effective syntax for FO queries with a bounded rewriting.

Finally, we extend bounded evaluability to graph pattern queries, by extending access constraints to graph data. We characterize bounded evaluability for subgraph and simulation patterns and develop practical algorithms for associated problems.

Lay Summary

Query answering is expensive on big data. A linear scan of a big dataset D may take days with a solid state device if D is of PB size and years if D is of EB size. In other words, conventionally efficient algorithms for query evaluation are already not feasible on big data. To tackle this, we propose querying big data with bounded data access, such that the evaluation cost is independent of the scale of D.

We first define *boundedly evaluable* queries under a set of indices built *w.r.t.* cardinality constraints over values in the datasets, such that for *any* dataset D satisfying the constraints, we can compute the exact answers to such queries in D by accessing a subset D_Q of D with the size of D_Q *independent* of the scale of D. We study the problem of checking whether queries are boundedly evaluable and develop an *effective syntax* of boundedly evaluable queries to reduce the complexity without sacrificing the expressive power of such queries. The effective syntax allows us to focus on queries of a simple normal form while still preserving the full power of bounded evaluability.

We then extend bounded evaluability to resource-bounded approximation and bounded rewriting using views, such that more queries can be answered with bounded resources (*e.g.,* time or space), exactly or approximately. Resource-bounded approximation allows the user to specify a resource ratio $\alpha \in (0,1]$ such that, for any dataset D and query Q, approximate answers with provable accuracy to Q in D can be computed by accessing only an α-fraction of D. Bounded rewriting using views incorporates cached historical query answers as materialized views so that queries can be answered exactly by accessing a bounded subset of D and the cached views only.

We also study bounded evaluability for graph pattern queries which are commonly found in *e.g.,* social network analysis and web data mining, so that we can identify cases when graph pattern matching can be done scale independently and develop algorithms that actually carry out the computation with bounded access only.

These techniques give us a framework of answering queries with bounded data access. Given a query Q, we first check whether it is boundedly evaluable. If so, we compute the exact answers with bounded data access. Otherwise, we check whether there are cached views that can make Q bounded; or we compute approximate answers with bounded data access if approximation is also desirable. If none of these works, we then directly evaluate Q using conventional approaches.

Acknowledgements

Above all, my deepest graditude is due to my supervisors, Professor Wenfei Fan at the University of Edinburgh and Professor Jinpeng Huai at Beihang University. Deeply thank Wenfei for his invaluable support, guidance, inspiration, patience, dedication and trust. He made it possible for me to study at Edinburgh, introduced me to the database area, trained me to have the right taste of research, and covered my back every time when I had difficultes. Jinpeng led me to the field of computer science, provided huge support from all aspects since I was an undergraduate, gave me dedicated guidance and inspiration, and was always available for help even though he was so busy due to his high-profile administrative position. Without the support of my two supervisors, I would not be possible to enroll in this joint Ph.D program between the two universities. I feel so lucky to have two extremely supportive supervisors.

Thanks to Professor Peter Buneman for building the database group at Edinburgh and creating such an enjoyable environment for research. Its diversity, quality and activity in research make it a wonderful place to study.

My special thanks go to Professor Don Sannella. Without his support at the Edinburgh side, the joint Ph.D program would not be possible.

I also accumulated debts of gratitude to my collaborators, including Dr. Ting Deng, Professor Floris Geerts, Ruizhe Huang, Dr. Ping Lu, Professor Shuai Ma, and Dr. Wenyuan Yu. In particular, my deeply-felt gratitude and thanks go to Floris and Ping, for their deep insights and for spending a lot of time to discuss questions with me. They directly contributed to a number of proofs in this work. I also thank Shuai and Wenyuan for working together in the early days of my PhD study, which is very important for my development. Thanks must also go to my colleagues, Chao Tian, Dr. Xin Wang, Yanghao Wang and Jingbo Xu, for their extended support. Especially thank Chao for taking care of everything about renting the flat, and Xin for many helpful suggestions in the preparation of this dissertation.

My sincere thanks go to Professor Peter Buneman and Professor Floris Geerts, for taking time to read my dissertation and serving as my examiners. Their constructive and insightful suggestions help improving and finalizing this thesis greatly. I feel really honored to have them on my examination committee.

Finally, I would like to sincerely thank my family: my parents for their unwavering and unconditional support over the years; my sister for her encouragements and selfless support; and my wife for her remarkable patience and reassuring love.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Yang Cao*)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this dissertation we develop a variety of methods for querying big data by accessing a bounded amount of small data, in order to deal with the unprecedented quantity of big data with limited resources.

In this chapter, we present the motivation for the study (Section 1.1), describe the main results of the work (Section 1.2), discuss related research (Section 1.3) and list related publications (Section 1.4).

1.1 Motivation

Querying big data is cost-prohibitive. On one hand, query answering is expensive in terms of complexity. Given a query Q and a dataset D, it is NP-complete to decide whether a tuple t is in the query answer $Q(D)$ when Q is an SPC query (selection, projection and Cartesian product), and PSPACE-complete if Q is in relational algebra (RA) [AHV95]. On the other hand, when D is big, even simple algorithms are not feasible in practice. Indeed, a linear scan of a dataset D of PB size (10^{15} bytes) takes days using a solid state drive with a reading speed of 6GB/s, and it takes years if D is of EB size (10^{18} bytes) [FGN13].

This motivates us to ask the following question: is it possible to compute $Q(D)$ by only accessing a bounded fraction D_Q of D that suffices to answer Q in D? More specifically, we want to know whether a query Q has the following *bounded evaluability* property: for *all* datasets D, there exists a subset $D_Q \subseteq D$ such that

- $Q(D_Q) = Q(D)$,
- the time for identifying D_Q by possibly using access information of D, and hence the size $|D_Q|$ of D_Q, are *independent of* $|D|$.

If a query Q satisfies these two conditions, we say that Q is *boundedly evaluable*. The need for studying boundedly evaluable queries is evident in practice. For a boundedly evaluable query Q, we can find a bounded dataset D_Q and then compute $Q(D)$ by using D_Q, *independent of* the possibly big D. Moreover, when D grows, the performance does not degrade. In other words, we can *reduce* big D to a “small” D_Q of a manageable size, with *bounded data access*.

While bounded evaluability is desirable, it is obviously not easy to achieve. However, real-life data often carries rich data semantics that can be utilized to make bounded evaluability feasible. To illustrate this, consider the following example.

Example 1: Consider a dataset D_0 of all traffic accidents in the UK from 1979 to 2005 [Gova], and a query Q_0 to find the ages of drivers who were involved in an accident in Queen’s Park district on May 1, 2005. The query is defined on three (simplified) relations Accident(aid, district, date), Casualty(cid, aid, class, vid) and Vehicle(vid, driver, age), recording accidents (where and when), casualties (class and vehicle), and vehicles (including driver information such as age), respectively. Query Q_0 is written as

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

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

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

The first two constraints state that from 1979 to 2005, at most 610 accidents happened within a single day, and each accident involved at most 192 vehicles, respectively. Constraint ψ_3 says that aid is a key for Accident; similarly for ψ_4. The constraints are discovered by simple aggregate queries on D_0. Indices can be built on D_0 based on ψ_1 such that given a date, it returns all ids (at most 610) of accidents that happened on the day; similarly for ψ_2–ψ_4. We refer to the cardinality constraints and their indices put together as *access constraints*.

Given the set $\mathcal{A}_0$ of the above access constraints, we can compute $Q_0(D_0)$ by accessing at most 234850 tuples from D_0, instead of millions. (1) We identify and fetch at most 610 aid's of Accident tuples with date = "1/5/2005", using the index built on ψ_1. (2) For each aid, we fetch its Accident tuple using the index for ψ_3. We select a set T_1 of tuples with district = "Queen's Park". (3) For each tuple $t \in T_1$, we fetch a set T_2 of at most 192 vid's from Casualty tuples with aid $= t[\text{aid}]$, via the index for ψ_2. (4) For each $s \in T_2$, we find a Vehicle tuple with vid = $s[\text{vid}]$, using the index for ψ_4. These tuples suffice for computing $Q_0(D_0)$, 610 + 610 $\times$ 192 $\times$ 2 in total, all fetched using indices. In fact, the chances are that we need to access 610 + 610 $\times$ 2 $\times$ 2 = 3050 tuples only, since accidents involved two vehicles on average. Better still, no matter how big D_0 grows, as long as D_0 satisfies ψ_1–ψ_4 (possibly with cardinality bounds mildly adjusted), $Q_0(D_0)$ can be computed by accessing a small number of tuples determined by Q_0 and the bounds in ψ_1–ψ_4 only. Thus Q_0 is *boundedly evaluable under access constraints* ψ_1–ψ_4. □

This example shows that, by combining index construction with the semantic constraints on datasets, it is possible to query big data by accessing bounded small data. To make practical use of the idea, several questions listed below have to be settled.

Questions. To make full use of boundedly evaluable queries in querying big data, we need to answer the following questions.

(**Q1**) Given a query Q and a set $\mathcal{A}$ of access constraints, can we decide whether Q is boundedly evaluable under $\mathcal{A}$? We know that this problem is undecidable for first-order logic (FO, the full relational algebra) queries in the absence of access constraints [FGL14], due to the undecidability of FO query satisfiability. Is it decidable for practical fragments of FO?

The problem is nontrivial. The first challenge is to formalize bounded evaluability under access constraints, *i.e.,* how to define that a query is boundedly evaluable. We need to specify how query evaluation can access data for *identifying* D_Q from D under access constraints in the first place, so that we can quantify data access for query evaluation. For instance, we need to define how query plans fetch aid's of Accident tuples with Q and access constraint ψ_1 in Example 1. Second, to characterize bounded evaluability, we need to reason about query equivalence *w.r.t.* the "customized" query evaluation under access constraints. This is more involved than the conventional query equivalence as here we consider the equality of query answers over a subset of database instances satisfying the cardinality constraints.

(**Q2**) The undecidability for FO hinders the use of boundedly evaluable queries in practice. How can we break the barrier of undecidability to make practical use of the idea? A typical approach is to identify decidable subclasses of boundedly evaluable FO queries. However, we do not want to sacrifice the expressive power when settling with these subclasses of boundedly evaluable queries.

(**Q3**) When a query Q is known to be boundedly evaluable under a set $\mathcal{A}$ of access constraints, how can we generate a query plan that can actually access D_Q for each dataset D satisfying $\mathcal{A}$, such that the size $|D_Q|$ of D_Q is bounded by Q and $\mathcal{A}$, and all answers to Q in D are preserved in D_Q? Does this process require a re-implementation of the conventional DBMS query engine?

(**Q4**) How can we handle queries that are not boundedly evaluable? Can we settle with approximate answers while ensuring the accessed data is still within the resources available? If so, how can we quantify the trade-off between the quality of approximate answers and the amount of data accessed, while we cannot access the entire dataset? In addition, do access constraints of the form in Example 1 suffice for such resource-bounded approximation? Is this extension of bounded evaluability generic enough to handle all queries that are not covered by bounded evaluability?

(**Q5**) Can we incorporate bounded evaluability with views? Materialized views are commonly used in practice. It would be desirable to make use of views such that queries can be answered by using the cached views and accessing only a bounded number of tuples in the database D under access constraints. In the presence of views, bounded evaluability analysis becomes more involved. Indeed, views may have unbounded size. They cannot be directly used for fetching data with access constraints as it may cause unbounded data access. However, a sub-plan with multiple such unbounded views may have output of bounded size, which can then be safely used to fetch data by further interacting with access constraints with unbounded data access.

(**Q6**) Can we extend bounded evaluability from relational queries to graph pattern queries? Big data graphs are commonly found in real-life, *e.g.,* web graphs and social networks. Querying such data graphs is typically done by graph pattern matching queries. Can we query big graphs with boundedly evaluable graph patten queries?

This may require an extension of access constraints for graph data and a redevelopment of bounded evaluability *w.r.t.* the semantics of graph pattern matching accordingly. In particular, it cannot be done by simply storing graph data in a relation of edges and expressing graph pattern matching queries (*e.g.,* subgraph isomorphism queries) by relational queries (*e.g.,* conjunctive queries) over the relations. This is because, while the queries can be expressed over the schema of the relations, the counterpart of access constraints on graph data cannot be expressed by simply using the relation schemas. Moreover, due to the difference on the express power, the bounded evaluability analysis for generic relational queries may be more expensive for graph queries, and thus cannot simply carry over to the latter. In addition, graph pattern matching by graph simulation needs recursions and is already beyond the scope of FO queries. Therefore, a native notion of bounded evaluability over graph queries on graph data is needed.

1.2 Contributions and Organization

In this dissertation, the following contributions are made to address the questions raised above. They are organized in three parts, which together provide a toolkit for querying big data with bounded data access.

Part I: Boundedly Evaluable Relational Queries

(1) Characterizing bounded evaluability. We answer question **Q1** in Chapter 2.

- We first formalize and characterize boundedly evaluable queries under access constraints. We extend conventional relational algebra query plans to incorporate access constraints and to quantify data access of query evaluation.
- We then study the *bounded evaluability problem*, denoted by BEP. Given a query Q and a set $\mathcal{A}$ of access constraints, BEP is to decide whether Q is boundedly evaluable under $\mathcal{A}$. Intuitively, it is to determine whether it is feasible to compute exact answers to Q in big datasets D by accessing a bounded amount of data from D, via boundedly evaluable query plans. It is known that BEP is undecidable for FO queries without access constraints [FGL14]. We show that, in the presence of access constraints, while it is already EXPSPACE-hard for CQ (*i.e.,* SPC), it is decidable for positive fragments of FO, *i.e.,*CQ, unions of conjunctive queries (UCQ, *i.e.,* SPCU), positive FO queries ($\exists$FO$^+$, select-project-join-union queries): they are all decidable in 2EXPSPACE.
  The upper bound proof is nontrivial and is based on a number of characterizations on bounded evaluability and query containment (equivalence) of CQ and UCQ under access constraints. The upper and lower bound to BEP do not match and it is an open problem to find a matching upper and lower bound for BEP.

(2) An effective syntax for boundedly evaluable queries. The undecidability bounded evaluability analysis for FO hinders the applicability of bounded evaluability in practice, as questioned in **Q2**. We approach this question by developing effective syntax for boundedly evaluable RA (full relational algebra) queries in Chapter 3.

- Under a set $\mathcal{A}$ of access constraints, we define a class $\mathcal{L}_{BE}(\mathcal{A})$ of RA queries as an effective syntax for bounded evaluability, referred to as queries *covered by* $\mathcal{A}$, such that
  (a) every boundedly evaluable RA query is equivalent to a query covered by $\mathcal{A}$, *i.e.,* the class $\mathcal{L}_{BE}(\mathcal{A})$ expresses all boundedly queries RA under $\mathcal{A}$;
  (b) every covered query is boundedly evaluable under $\mathcal{A}$; and
  (c) it takes PTIME in $|Q|$ and $|\mathcal{A}|$ to check whether Q is covered by $\mathcal{A}$.

  Intuitively, covered queries make a *core* subclass of boundedly evaluable RA queries under $\mathcal{A}$, *without sacrificing their expressive power.*
- Capitalizing on the effective syntax, we develop a bounded evaluation framework for querying with bounded evaluability, as an answer to **Q3**. Under a set $\mathcal{A}$ of access constraints, given an input RA query Q,
  - the framework first checks whether Q is covered by $\mathcal{A}$ in PTIME by condition (c) above;
  - if so, it generates a bounded query plan for Q by using indices in $\mathcal{A}$, which is waranted to exist by condition (b) above and can be directly executed on top of DBMS;
  - Otherwise, it evaluates Q directly using conventional DBMS query engine.

  By condition (a), if Q is boundedly evaluable, it can always be expressed in $\mathcal{L}_{BE}(\mathcal{A})$ and can be effectively answered using the framework.
- We develop algorithms underlying the framework to check the coverage of Q, and to generate a bounded query plan for covered Q. We also study optimization problems to minimize data access of bounded query plans.

Part II: Beyond Boundedly Evaluable Queries

(3) Extending bounded evaluability to bounded approximation. We answer question **Q4** in Chapter 4.

- We extend bounded evaluability to a resource bounded approximation scheme, so that we can handle queries that are not boundedly evaluable. It is parameterized with a *resource ratio* $\alpha \in (0,1]$, indicating that our available resources allow us to only access an α- fraction of a big dataset D. Underlying the scheme are

(a) access templates that extend access constraints for approximation on any database instances; and

(b) an accuracy measure to assess approximate answers.

Given any RA query Q and instance D of relational schema $\mathcal{R}$ that satisfies $\mathcal{A}$, the scheme is to find a set S of tuples and a provable deterministic accuracy bound η such that

- it accesses a fraction D_Q of D with $|D_Q| \leqslant \alpha|D|$, and
- the accuracy of S to Q in D is at least η.

- We show that, for any relational schema $\mathcal{R}$, we can always extend a set $\mathcal{A}$ of access constraints with access templates, denoted by $\mathcal{A}'$, such that for any instance D of $\mathcal{R}$ that satisfies $\mathcal{A}$, any resource ratio $\alpha \in (0,1]$ and any RA query Q, we can compute approximate answers S to Q in D and an accuracy bound η, by accessing D_Q via $\mathcal{A}'$, where $|D_Q| \leqslant \alpha|D|$, such that the accuracy of S to Q in D is no smaller than η. We develop resource-bounded approximation algorithms for SPC, RA and aggregate RA queries as constructive proofs.

- On top of DBMS, we implement a resource-bounded framework that combines the bounded approximation scheme and bounded evaluation in (3) above, for answering RA queries with bounded resource. Given a set $\mathcal{A}'$ of access constraints and templates, a resource ratio α, and an instance D of $\mathcal{R}$ that satisfies $\mathcal{A}'$, and an RA query Q,

  (a) it checks whether Q is boundedly evaluable under $\mathcal{A}'$;

  (b) if so, it computes $Q(D)$ by accessing bounded $D_Q \subseteq D$;

  (c) otherwise, it identifies D_Q with $|D_Q| \leqslant \alpha|D|$ by using $\mathcal{A}'$, and computes $Q(D_Q)$ along with a deterministic bound η based on the bounded approximation scheme.

(4) Bounded evaluability with views. To tackle question **Q5**, in Chapter 5 we study bounded query rewriting using view, which extends bounded evaluability to incorporate materialized views.

- We first formalize bounded query rewriting in the presence of access constraints and views. A query Q has a bounded rewriting using a set $\mathcal{V}$ of views under a set $\mathcal{A}$ of access constraints, if there exists a query Q' expressed in the same language as Q, such that given any dataset D satisfying $\mathcal{A}$, $Q(D)$ can be computed by Q' that accesses only cached views and additionally, a bounded fraction D_Q of D that can be identified within time independent of the size $|D|$ of D (and hence its size $|D_Q|$). Here we only restrict the size of data access in D as although $\mathcal{V}(D)$ may not be bounded, we often use small views cached with fast access [ALK$^+$13]. We formalize the notion by extending bounded query plans with views and a bounded size M determined by our available resources such as processors and time constraint. Note that, in the absence of M, the problem is already EXPSPACE-hard for CQ as shown in Part I(1).

- We then study the problem of deciding whether a query has a bounded rewriting with a set of views under a set of access constraints, denoted by VBRP. We study VBRP($\mathcal{L}$) when $\mathcal{L}$ ranges over CQ, UCQ and FO queries. We show that VBRP is Σ_3^p-complete for CQ, UCQ and $\exists$FO$^+$; but it becomes undecidable for FO.
- To explore effective use of bounded evaluability with views, We also develop effective syntax for queries that have a bounded rewriting, analogous to (2) above.

Part III: Bounded Evaluability on Graphs

(5) Bounded evaluability on graph pattern queries. Finally, we extend the study of bounded evaluability to graph pattern queries in Chapter 6, to tackle question **Q6**.

- We first extend bounded evaluability to graph pattern queries. We formulate access constraints on graphs and define boundedly evaluable pattern queries.
- We then characterize boundedly evaluable *subgraph queries* Q, *i.e.,* patterns defined by subgraph isomorphism. We develop a characterization for checking whether Q is boundedly evaluable under a set $\mathcal{A}$ of access constraints. Based on it, we develop cubic time algorithms for checking whether Q is bounded under $\mathcal{A}$, and generating bounded plans for Q if it is boundedly evaluable.
- If Q is not bounded under $\mathcal{A}$, we propose to make it *instance-bounded*. That is, for a given graph G that satisfies $\mathcal{A}$, we find an extension $\mathcal{A}_M$ of $\mathcal{A}$ satisfied by G such that under $\mathcal{A}_M$, we can find $G_Q \subset G$ in time decided by $\mathcal{A}_M$ and Q, and $Q(G_Q) = Q(G)$. We show that when the size of indices in $\mathcal{A}_M$ is constrained, the problem for deciding the existence of $\mathcal{A}_M$ is in low polynomial time (PTIME).
- We also extend the study to *simulation queries*, *i.e.,* patterns interpreted by graph simulation, where we need to cope with the *non-localized* and *recursive* nature

of graph simulation.

1.3 Related Research

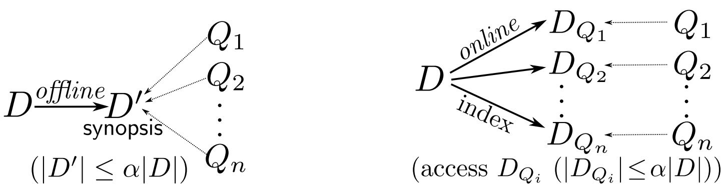
Scale independence. The study of bounded evaluability is motivated by the idea of scale independence [AFP$^+$09]. The latter aims to guarantee that a bounded amount of work is required to execute all queries in an application, regardless of the size of the underlying data. To enforce scale independence, users may specify bounds on the amount of data accessed and the size of intermediate results; when more data is needed, only top-k tuples are retrieved to meet the bounds [ACK$^+$11].

The idea was formalized in [FGL14]. A query Q is called *scale independent* in a dataset D *w.r.t.* a bound M if there is $D_Q \subseteq D$ such that $Q(D) = Q(D_Q)$ and $|D_Q| \leqslant M$. Access constraints were introduced in [FGL14]. A notion of $\bar{x}$-scale independence was also proposed in [FGL14], to characterize queries $Q(\bar{x}, \bar{y})$ that, for all databases D that satisfy access constraints and for each tuple $\bar{a}$ of values for $\bar{x}$, $Q(\bar{x} = \bar{a}, D)$ can be computed in time dependent on $\mathcal{A}$ and Q only. It was shown that x-scale independence is undecidable for FO, and syntactic rules were developed as a sufficient condition for deciding the x-scale independence of FO queries under access constraints.

This work differs from the prior work as follows. (1) While [FGL14] has mostly focused on scale independence in *a given database*, we focus on bounded evaluability on *all databases* that satisfy access constraints, like x-scale independence. This leads to more intriguing analysis. For instance, it is EXPSPACE-hard to decide whether a CQ query is boundedly evaluable, in contrast to Σ_3^p-complete [FGL14]. (2) We give characterizations for bounded evaluability of queries for various fragments of FO, and show that they are decidable. (3) We also study effective syntax and develop practical algorithms based on the syntax, which are not studied before.

Access constraints. Related to access constraints is the notion of access patterns. Access patterns require that a relation can only be accessed by providing certain combinations of attribute values. Query processing under limited access patterns has been well studied, *e.g.,* [BBB13, DLN07, Li03, NL04]. In contrast, access constraints combine indices and cardinality constraints. Our goal is to characterize what queries are boundedly evaluable with access constraint, rather than to study the complexity or executable plans for answering queries under access patterns [BBB13, DLN07, Li03, NL04].

Effective syntax. The notion of effective syntax was first studied by M.Y. Vardi [Var81]



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

Figure 1.1: Data reduction schemes

and J.D. Ullman ([Ull82]). Since then it has been developed in many areas, *e.g.,* safe relational queries [GT91] and finite queries [ST95]. The idea is that while a query class $\mathcal{Q}$ is undecidable or has high complexity, it may be possible to impose syntactical restrictions on $\mathcal{Q}$ such that the restricted subclass is decidable and has a low complexity, and moreover, every query in $\mathcal{Q}$ is equivalent to one in the subclass. We develop effective syntax for various fragments of FO (RA) queries. This makes the idea of bounded evaluability applicable in practice.

Approximate query answering. Related to resource bounded approximation is approximation query answering. Prior work on approximate query answering is based on either (a) synopsis [HHW97, BCD03a, IP99, JKM+09, Dob05, CGRS01, CG05], or (b) views, such as dynamic sampling [BCD03b] and BlinkDB [AMP+13] (see [CGHJ12] for a survey). 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 [AMP+13]. 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 aggregate queries by using the samples instead of D, with probabilistic error rates.

Our resource-bounded approximation scheme radically differs from the prior work in the following. (1) As shown in Fig. 1.1, our approximation scheme is based on *dynamic data reduction* that identifies D_Q for each input query Q, as opposed to an *one-size-fit-all* synopsis D' [HHW97, BCD03a, IP99, JKM+09, Dob05, CGRS01, CG05], and to pre-computed views [AMP+13, CGN15]. This allows us to get a provable accuracy bound and employ access constraints and templates for data reduction with bounded resource. (2) Prior approaches "substantially limit the types of queries they can execute" [AMP+13], and often focus on aggregate queries. Our scheme works on generic queries, aggregate or not. (3) Previous methods make various assumptions on future queries, *i.e.,* workloads, query predicates or QCSs are known in advance [AMP+13, CGN15]. For *unpredictable queries*, however, traditional techniques "can do little more than rely on query optimizers for individual queries" [AMP+13]. Moreover, they provide either no accuracy guarantee at all, or probabilistic/statistical error rates for aggregate queries. Such error rates do not tell us how "good" each approximate answer is. We target unpredictable queries *without* any assumption on workloads or QCSs, and guarantee *deterministic accuracy* on each answer.

Query rewriting using views. Query rewriting has been extensively studied (*e.g.,* [LMSS95, Afr11, ALU07, RSU95, CNS99, NSV10]; see [Len02, Hal01] for surveys). In contrast to conventional query rewriting using views, bounded rewriting allows controlled access to the underlying dataset D, in addition to cached views $\mathcal{V}(D)$, under access constraints. This makes the rewriting analysis more challenging. For instance, it is Σ_3^p-complete to decide whether there exists a bounded rewriting for CQ with CQ views, as opposed to NP-complete in the conventional setting [LMSS95].

More discussions on related research in query optimization (*e.g.,* column-stores [AMH08] and index-only scans [Pos]) are given in the end of Chapter 3, after more technical details on bounded evaluability and access constraints are given.

1.4 List of Publications

During the course of the PhD study at the University of Edinburgh, I have published the following publications as a co-author that are relevant to this thesis.

- [CF16] **Yang Cao** and Wenfei Fan. An effective syntax for bounded relational queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD*, 2016.
- [CFGL16] **Yang Cao**, Wenfei Fan, Floris Geerts, and Ping Lu. Bounded query rewriting using views. In *Proceedings of the 35th ACM Symposium on Principles of Database Systems, PODS*, 2016.
- [FGC+15] Wenfei Fan, Floris Geerts, **Yang Cao**, Ting Deng, and Ping Lu. Querying big data by accessing small data. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS*, 2015.
- [CFHH15] **Yang Cao**, Wenfei Fan, Jinpeng Huai, and Ruizhe Huang. Making pattern queries bounded in big graphs. In *31st IEEE International Conference on Data Engineering, ICDE*, 2015.

- [CFWY14] **Yang Cao**, Wenfei Fan, Tianyu Wo, and Wenyuan Yu. Bounded conjunctive queries. *PVLDB*, 7(12), 2014.

Remark. Part of the results in this dissertation appear in the above publications: (1) Partial results in Chapter 2 have been previously published in PODS'15 [FGC$^+$15], but some major definitions and the main result, *i.e.,* the 2EXPSPACE upper bound proof, have been redeveloped and are presented here for the first time; (2) Most of the results in Chapter 3 have been previously published in SIGMOD'16 [CF16]; (3) Part of the results in Chapter 5 have been previously published in PODS'16 [CFGL16], with some new results regarding the effective syntax part. (4) Most of the results in Chapter 6 have been previously published in ICDE'15 [CFH15]. Part of the results in Chapter 4 are taken from a recently finished paper under submission.

Part I

Boundedly Evaluable Relational Queries

In Part I, we investigate the feasibility of querying big data by accessing a bounded amount of the data via bounded evaluability.

In Chpater 2, we formulate and study boundedly evaluable relational queries under a set $\mathcal{A}$ of access constraints (called an *access schema*), such that the evaluation cost of such queries is determined by the queries and constraints only, *independent of the scale of D*. We study the complexity of deciding bounded evaluability for several fragments of FO queries. We show that, while it is undecidable for FO without access constraints, bounded evaluability is decidable in $\mathsf{2EXPSPACE}$ for positive fragments of FO (*i.e.,* CQ, UCQ and $\exists\mathsf{FO}^+$) in the presence of access constraints, but is already $\mathsf{EXPSPACE}$-hard for CQ.

In light of the undecidability of bounded evaluability analysis, in Chapter 3, we explore effective syntax to make practical use of bounded evaluability. Under an access schema $\mathcal{A}$, an effective syntax is a class of queries such that (a) every query that is boundedly evaluable under $\mathcal{A}$ must be equivalent to one in the class; (b) every query in the class is boundedly evaluable under $\mathcal{A}$; and (c) it is in PTIME to decide whether a query is in the class. Capitalizing on the effective syntax, we develop a bounded evaluation framework on top of existing DBMS, and develop practical algorithms underlying the framework. We also conduct experiments to verify the effectiveness of bounded evaluability and the bounded evaluation framework on real-life datasets.

Chapter 2

Bounded Evaluability

This chapter studies bounded evaluability for relational queries under access constraints and investigates its feasibility and complexity. While it is undecidable to determine whether FO queries are boundedly evaluable, we show that for several classes of FO queries, the bounded evaluability problem is decidable.

Below we first define access constraints and bounded evaluability in Section 2.1, and then characterize bounded evaluability for several classes of queries in Section 2.2.

2.1 Boundedly Evaluable Queries

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

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

Query classes. We study the following queries [AHV95].

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

If $\bar{x}$ is the tuple of free variables of Q, we will write $Q(\bar{x})$. Given a query $Q(\bar{x})$ with $|\bar{x}| = m$, the answer to Q in D, denoted by $Q(D)$, is the set $\{\bar{a} \in \mathsf{adom}(D)^m \mid D \models Q(\bar{a})\}$, where the active domain, $\mathsf{adom}(D)$, consists of all constants appearing in D or Q.

We consider CQ queries that are either constant queries, *e.g.,* $x = 1$, or queries where all variables occur at least once in relation atoms. We also assume *w.l.o.g.* that CQ queries are safe and satisfiable, *i.e.,* each variable is equal to either a variable occurring in a relation atom or a constant, and can be equal to at most one constant. For FO queries, we also consider safe queries [AHV95].

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

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

where R is a relation schema in $\mathcal{R}$, X and Y are sets of attributes of R, and N is a natural number. A relation instance D of R *satisfies* the constraint if

- for any X-value $\bar{a}$ in D, $|D_Y(X = \bar{a})| \leqslant N$, where $D_Y(X = \bar{a}) = \{t[Y] \mid t \in D,\ t[X] = \bar{a}\}$; and

- there exists an *index on X for Y* that given an X-value $\bar{a}$, retrieves $D_Y(X = \bar{a})$ in $O(N)$ time.

For instance, ψ_1–ψ_4 given in Example 1 together with their indices are access constraints. An access constraint is a combination of a cardinality constraint and an index on X for Y. It tells us that given any X-value, there exist at most N distinct corresponding Y-values, and these Y values can be retrieved by using the index. say that D *satisfies* access schema $\mathcal{A}$, denoted by $D \models \mathcal{A}$, if D satisfies all the constraints in $\mathcal{A}$.

Remark. (1) Access constraints can be automatically discovered by extending mining algorithms for functional dependencies (FDs), *e.g.,* [HKPT99], with simple aggregate queries. (2) Traditional FDs are a special case of access constraints, when $N = 1$. (4) A more restricted form of access constraints is $R(X \to X, 1)$, *e.g.,* ψ_3 of Example 1, referred to as *indexing constraints*. Such constraints simply enforce a (hash) index on attributes X of R; in the presence of the index, all instances of R satisfy the constraints.

Query plans. To define boundedly evaluable queries, we first present query plans. Consider a query Q in the relational algebra over schema $\mathcal{R}$, defined in terms of projection operator π, selection σ, Cartesian product $\times$, union $\cup$, set difference $\setminus$ and renaming ρ (see, *e.g.,* [AHV95] for details). A *query plan* for Q is a sequence

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

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

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

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

We say that a query plan $\xi(Q, \mathcal{R})$ is *boundedly evaluable under an access schema* $\mathcal{A}$ if (1) for each operation $\mathsf{fetch}(X \in T_j, R, Y)$ in it, there exists a constraint $R(X \to Y', N)$ in $\mathcal{A}$ such that $Y \subseteq X \cup Y'$, and 2) the length of $\xi(Q, \mathcal{R})$ (*i.e.,* the number of operations) is determined only by $|\mathcal{R}|$, $|\mathcal{A}|$ and $|Q|$ which are the sizes of $\mathcal{R}$, $\mathcal{A}$ and Q, respectively, *independent of* the size $|D|$ of dataset D.

Intuitively, if $\xi(Q,\mathcal{R})$ is boundedly evaluable under $\mathcal{A}$, then for all instances D of $\mathcal{R}$ that satisfy $\mathcal{A}$, $\xi(Q,\mathcal{R})$ tells us how to fetch $D_Q \subseteq D$ by using the indices in $\mathcal{A}$ such that $Q(D) = Q(D_Q)$, where D_Q is the set of all tuples fetched from D by following $\xi(Q,\mathcal{R})$. Better still, D_Q is *bounded*: $|D_Q|$ is determined by Q and constants in $\mathcal{A}$ only. Moreover, the time for identifying and fetching D_Q also depends on Q and $\mathcal{A}$ only (assuming that given an X-value $\bar{a}$, it takes $O(N)$ time to fetch $D_{XY}(X = \bar{a})$ in D with the index in $R(X \to Y,N)$). For instance, a boundedly evaluable query plan for Q_0 is outlined in Example 1 under access constraints ψ_1–ψ_4.

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

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

One can readily verify the following: if Q is boundedly evaluable under $\mathcal{A}$, then for *all* instances D of $\mathcal{R}$ that satisfy $\mathcal{A}$, there exists $D_Q \subseteq D$ such that (a) $Q(D_Q) = Q(D)$; and (b) the time for identifying and fetching D_Q is determined by Q and $\mathcal{A}$, *not* by the size $|D|$ of D; and the size $|D_Q|$ is also *independent of* $|D|$.

2.2 Deciding Bounded Evaluability

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

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

While BEP(FO) is undecidable even $\mathcal{A} = \emptyset$ ([FGL14]), we show that for several practical fragments of FO, BEP is decidable.

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

Example 2: (1) Consider an access schema $\mathcal{A}_1$ and a query Q_1 defined over a relation schema $R_1(A,B,E,F)$:

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

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

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

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

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

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

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

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

$$Q_3'(x,x) = R_3(1,1,x) \wedge R_3(x,x,x).$$

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

To see that Q_3 is "$\mathcal{A}_3$-equivalent" to Q_3', observe the following: (a) by φ_4, x, y and z_3 must take the same (unique) value c_0 from D_3, which can be fetched by using the index built for φ_4; hence $R(x,y,z_3)$ becomes $R(x,x,x)$; and (b) $\exists z_1, z_2(R(1,1,x) \wedge R(z_1,z_2,y))$

is equivalent to $R(1,1,x)$; thus $R(z_1,z_2,y)$ can be removed. Moreover, Q_3' is boundedly evaluable under $\mathcal{A}_3$ since by φ_5, we can check whether $(1,1,x)$ and (x,x,x) are in D_3 when $x = c_0$, using the index for φ_5. □

The example tells us that to decide whether a CQ Q is boundedly evaluable under an access schema $\mathcal{A}$, we need to find out (a) whether Q is "$\mathcal{A}$-equivalent" to a query Q' that is boundedly evaluable under $\mathcal{A}$, or (b) whether the indices in $\mathcal{A}$ "cover" attributes corresponding to variables in Q. Below we formalize this intuition.

2.2.1 Impact of Access Constraints

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

When Q is a query in CQ, it is in PTIME to decide whether there exists D such that $Q(D) \neq \emptyset$ (cf. [AHV95]). In contrast, $\mathcal{A}$-satisfiability is intractable for CQ.

Lemma 1: *It is* NP*-complete to decide whether a query in* CQ *is $\mathcal{A}$-satisfiable for an access schema $\mathcal{A}$.* □

To prove this, we need the following notation. Consider a tableau (T_Q, u) representing a CQ Q. A valuation θ of (T_Q, u) is a mapping from variables in T_Q to (not necessarily distinct) constants in $\mathbf{U}$. We use $\theta(T_Q)$ to denote the instance obtained by applying θ to variables in T_Q. We call $(\theta(T_Q), \theta(u))$ an *$\mathcal{A}$-instance* of Q if $\theta(T_Q) \models \mathcal{A}$. There are possibly exponentially many $\mathcal{A}$-instances of Q up to isomorphism, analogous to representative instances in indefinite data [Klu88, KMT98, vdM97].

Proof: We show that it is NP-complete to decide whether a CQ Q is $\mathcal{A}$-satisfiable for an access schema $\mathcal{A}$.

Upper bound. We give an algorithm that, given an access schema $\mathcal{A}$ and a CQ Q, both defined over a relational schema $\mathcal{R}$, decides whether Q is $\mathcal{A}$-satisfiable, as follows:

(a) Guess a valuation θ for the tableau representation (T_Q, u) of Q. Here the valuation takes values from a finite domain $\mathbb{D}$ consisting of the constants appearing in Q and one constant a_x for each variable x in Q.
(b) If $\theta(T_Q) \models \mathcal{A}$ and $\theta(u)$ is defined (*i.e.,* there is no conflicts in $\theta(T_Q)$), then return "yes"; otherwise reject the guess and repeat steps (a) and (b).

Since step (b) is in PTIME, this is in NP. Clearly, when the algorithm returns "yes"

$$I_{01} = \begin{array}{|c|} X \\ \hline 1 \\ 0 \end{array} \quad I_{\vee} = \begin{array}{|ccc|} B & A_1 & A_2 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{array} \quad I_{\wedge} = \begin{array}{|ccc|} B & A_1 & A_2 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{array} \quad I_{\neg} = \begin{array}{|cc|} A & \bar{A} \\ \hline 0 & 1 \\ 1 & 0 \end{array}$$

Figure 2.1: Relation instances used in the lower bound proof of Lemma 1

then $\theta(T_Q)$ is an instance of $\mathcal{R}$ such that $Q(\theta(T_Q))$ is non-empty. In particular, $\theta(u)$ is in the query result. Conversely, if there exists an instance D of $\mathcal{R}$ such that $D \models \mathcal{A}$ and $Q(D)$ is non-empty, then there exists a valuation θ' of T_Q into D such that $\theta'(u) \in Q(D)$. It is readily verified that θ' can be turned into a valuation θ of T_Q that takes values from $\mathbb{D}$ and such that $\theta(u)$ is defined. Indeed, let $\mathbb{D}'$ be set of constants in $\theta'(T_Q)$. Then $|\mathbb{D}'| \leqslant |\mathbb{D}|$ and we can define an injective function $\iota : \mathbb{D}' \to \mathbb{D}$ such that $\iota(a) = a$ for every constant a in Q, and $\iota(b) = a_x$ for some unique constant $a_x \in \mathbb{D}$ otherwise. Then, $\theta = \iota \circ \theta'$ is the desired valuation that can be guessed by the algorithm. Hence, if Q is $\mathcal{A}$-satisfiable then the algorithm returns "yes".

Lower bound. We show that the problem is NP-hard by reduction from SAT. An instance of SAT is a propositional formula $\psi = C_1 \wedge \cdots \wedge C_r$ defined on variables $X_\psi = \{x_1, \ldots, x_m\}$, where for each $i \in [1,r]$, clause C_i is of the form $\ell_1^i \vee \ell_2^i \vee \ell_3^i$, and for each $j \in [1,3]$, literal ℓ_j^i is either a variable x_l in X_ψ or the negation of x_l. Given ψ, SAT is to decide whether there exists a truth assignment for X_ψ that satisfies ψ. It is NP-complete (cf. [Pap94]).

Given an instance ψ of SAT, we construct a relational schema $\mathcal{R}$, an access schema $\mathcal{A}$ over $\mathcal{R}$ and a CQ Q such that ψ is satisfiable iff Q is $\mathcal{A}$-satisfiable.

(a) The relational schema $\mathcal{R}$ consists of four relation schemas: $R_{01}(X)$, $R_{\vee}(B, A_1, A_2)$, $R_{\wedge}(B, A_1, A_2)$ and $R_{\neg}(A, \bar{A})$. Intuitively, R_{01} encodes the Boolean domain, and $R_{\vee}$, $R_{\wedge}$ and $R_{\neg}$ encode disjunction, conjunction and negation, respectively, such that ψ can be encoded in CQ in terms of these relations. Their instances I_{01}, $I_{\vee}$, $I_{\wedge}$ and $I_{\neg}$ are shown in Figure 2.1. As will be seen shortly, we use access constraints in $\mathcal{A}$ and query Q to ensure that I_{01}, $I_{\vee}$, $I_{\wedge}$ and $I_{\neg}$ are well defined.

(b) The access schema $\mathcal{A}$ consists of four access constraints on R_{01}, $R_{\vee}$, $R_{\wedge}$ and $R_{\neg}$ to ensure that the instances of these relation schemas indeed encode Boolean values and operations: $R_{01}(\emptyset \to X, 2)$, *i.e.,* I_{01} has at most two values; similarly, $R_{\vee}(\emptyset \to (B, A_1, A_2), 4)$, $R_{\wedge}(\emptyset \to (B, A_1, A_2), 4)$, and $R_{\neg}(\emptyset \to (A, \bar{A}), 2)$.

(c) We define query Q as

$$Q(u) = Q_c \wedge Q_{\psi}(u) \wedge u = 1,$$

where Q_c and Q_{ψ} are in CQ. Query Q_c is to ensure that instances I_{01}, $I_{\vee}$, $I_{\wedge}$ and $I_{\neg}$ of R_{01}, $R_{\vee}$, $R_{\wedge}$ and $R_{\neg}$, respectively, are precisely the relations shown in Figure 2.1. It is defined as $Q_{01} \wedge Q_{\vee} \wedge Q_{\wedge} \wedge Q_{\neg}$. For instance,

$$Q_{01} = \exists x_1, x_2 \big(R_{01}(x_1) \wedge x_1 = 0 \wedge R_{01}(x_2) \wedge x_2 = 1\big).$$

Together with constraint $R_{01}(\emptyset \to X, 2)$, it ensures that for any instance D of $\mathcal{R}$ that satisfies $\mathcal{A}$, if Q_{01} is nonempty, then the instance I_{01} of R_{01} in D is the one shown in Figure 2.1. We define $Q_{\vee}$, $Q_{\wedge}$ and $Q_{\neg}$ similarly.

Query Q_{ψ} encodes the SAT instance ψ, and is defined as follows:

$$Q_{\psi}(u) = \exists \bar{x} \big(\bigwedge_{i=1}^{m} R_{01}(x_i) \wedge Q'_{\psi}(u, \bar{x})\big),$$

where m is the number of variable in X_{ψ}, $\bigwedge_{i=1}^{m} R_{01}(x_i)$ selects a truth assignment μ_X for X_{ψ} from I_{01}, and $Q'_{\psi}(u, \bar{x})$ encodes the truth value of ψ for a given truth assignment μ_X for X_{ψ} such that $u = 1$ if ψ is satisfied by μ_X, and $u = 0$ otherwise. Query Q'_{ψ} can be expressed in CQ in terms of $R_{\vee}$, $R_{\wedge}$ and $R_{\neg}$. It includes a conjunction $\bigwedge_{j \in [1,r]} Q_j$, where for each $j \in [1, r]$, Q_j encodes clause C_j. For example, consider formula $\psi = C_1 \wedge C_2$, where $C_1 = x_1 \vee y_1 \vee \bar{z}_2$ and $C_2 = x_2 \vee \bar{y}_2 \vee z_1$. We encode clause C_1 by CQ query $Q_1(x_1, y_1, z_2, v_1) = \exists v'_1, z'_2 \ \big(R_{\vee}(v'_1, x_1, y_1) \wedge R_{\vee}(v_1, v'_1, z'_2) \wedge R_{\neg}(z_2, z'_2)\big)$. Query $Q_2(x_2, y_2, z_1, v_2)$ for C_2 is constructed similarly. Then query Q'_{ψ} is given by $Q_{\psi}(x_1, x_2, y_1, y_2, z_1, z_2, v) = \exists v_1, v_2, v' \ \big(Q_1(x_1, y_1, z_2, v_1) \wedge Q_2(x_2, y_2, z_1, v_2) \wedge R_{\wedge}(v', v_1, v_2)\big)$. Note that Q_{ψ} is nonempty on instances of $\mathcal{R}$ that satisfy $\mathcal{A}$ iff there exists a truth assignment μ_X of X_{ψ} that satisfies ψ.

We next show that ψ is satisfiable if and only if Q is Q is $\mathcal{A}$-satisfiable.

$\boxed{\Rightarrow}$ If ψ is satisfiable, then there exists a truth assignment μ_X for X_{ψ} that satisfies ψ. Hence there exists an instance D of $\mathcal{R}$ such that $D \models \mathcal{A}$ and $Q(D)$ is nonempty, *i.e.,* $Q_c(D)$ and $Q_{\psi}(D)$ are nonempty. Hence Q is $\mathcal{A}$-satisfiable.

$\boxed{\Leftarrow}$ Conversely, if ψ is not satisfiable, then there exists no truth assignment μ_X for X_ψ that satisfies ψ. Hence for any instance D of $\mathcal{R}$, if $D \models \mathcal{A}$, $Q'_\psi(D)$ is empty and thus $Q(D)$ is empty. That is, Q is not $\mathcal{A}$-satisfiable. □

Recall that query containment and equivalence are NP-complete for CQ, by the Homomorphism Theorem [CM77]. These no longer hold in the presence of an access schema $\mathcal{A}$. We say that a query Q_1 is *$\mathcal{A}$-contained* in query Q_2, denoted by $Q_1 \sqsubseteq_\mathcal{A} Q_2$, if for all instances D of $\mathcal{R}$ such that $D \models \mathcal{A}$, $Q_1(D) \subseteq Q_2(D)$. We say that Q_1 and Q_2 are *$\mathcal{A}$-equivalent*, denoted by $Q_1 \equiv_\mathcal{A} Q_2$, if $Q_1 \sqsubseteq_\mathcal{A} Q_2$ and $Q_2 \sqsubseteq_\mathcal{A} Q_1$. Then we have the following.

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

Proof: It suffices to show that it is Π_2^p-complete to decide whether $Q_1 \sqsubseteq_\mathcal{A} Q_2$. For if it holds, then it follows immediately that it is Π_2^p-complete to decide whether $Q_1 \equiv_\mathcal{A} Q_2$.

To show that it is Π_2^p-complete to decide whether $Q_1 \sqsubseteq_\mathcal{A} Q_2$, we first characterize $Q_1 \sqsubseteq_\mathcal{A} Q_2$, and then verify the Π_2^p bound. Consider Q_1 and Q_2 defined over a relational schema $\mathcal{R}$, and an access schema $\mathcal{A}$ over the same $\mathcal{R}$. Let (T_{Q_1}, u) be the tableau representation of Q_1.

(1) Characterization. We show that $Q_1 \sqsubseteq_\mathcal{A} Q_2$ if and only if (a) Q_1 is not $\mathcal{A}$-satisfiable; or (b) for any valid valuation θ of (T_{Q_1}, u) we have that $\theta(u) \in Q_2(\theta(T_{Q_1}))$. Here a valid valuation is one such that $\theta(T_{Q_1}) \models \mathcal{A}$ and $\theta(u)$ is defined.

First assume that $Q_1 \sqsubseteq_\mathcal{A} Q_2$. If there exists a valid valuation θ of T_{Q_1}, then $\theta(T_{Q_1}) \models \mathcal{A}$ and hence $Q_1(\theta(T_{Q_1})) \subseteq Q_2(\theta(T_{Q_1}))$. Since $\theta(u) \in Q_1(\theta(T_{Q_1}))$ we also have that $\theta(u) \in Q_2(\theta(T_{Q_1}))$.

Conversely, consider the following two cases. (a) If Q_1 is not $\mathcal{A}$-satisfiable, then obviously, $Q_1 \sqsubseteq_\mathcal{A} Q_2$. (b) If Q_1 is $\mathcal{A}$-satisfiable, assume that for all valid valuations θ of (T_{Q_1}, u), $\theta(u) \in Q_2(\theta(T_{Q_1}))$. Then for any instance D of $\mathcal{R}$, if $D \models \mathcal{A}$ and $Q(D)$ is non-empty then there must exist a valid valuation θ of (T_{Q_1}, u) with constants in D, such that $\theta(u) \in Q_1(\theta(T_{Q_1}))$. Observe that $(\theta(T_{Q_1}), \theta(u))$ is an $\mathcal{A}$-instance of $\mathcal{R}$. Hence $\theta(u) \in Q_2(\theta(T_{Q_1}))$. Since Q_2 is monotonic, $Q_2(\theta(T_{Q_1})) \subseteq Q_2(D)$. Hence $\theta(u) \in Q_2(D)$ and therefore, $Q_1 \sqsubseteq_\mathcal{A} Q_2$.

(2) Complexity. We next show that it is Π_2^p-complete to decide whether $Q_1 \sqsubseteq_\mathcal{A} Q_2$.

Upper bound. We give an Σ_2^p algorithm that, given an access schema $\mathcal{A}$ and CQ Q_1 and Q_2, all defined over a relational schema $\mathcal{R}$, checks whether $Q_1 \not\sqsubseteq_{\mathcal{A}} Q_2$.

(a) Check whether Q_1 is not $\mathcal{A}$-satisfiable. If so, return "no"; and continue otherwise.

(b) Guess an $\mathcal{A}$-instance $(\theta(T_Q), \theta(u))$ of Q_1. The valuation takes values from the domain $\mathbb{D}$ as defined in the proof of Lemma 1.

(c) Check whether $\theta(u) \notin Q_2(\theta(T_Q))$. If so, return "yes"; otherwise reject the guess and repeat steps (b) and (c).

The algorithm is in Σ_2^p since step (b) calls an NP oracle to check whether a tuple is an answer to a CQ in a database (see, *e.g.,* [AHV95]), and step (a) is in coNP . Hence the problem is in Π_2^p.

This algorithm correctly decides whether $Q_1 \not\sqsubseteq_{\mathcal{A}} Q_2$. When "yes" is returned then Q_1 is $\mathcal{A}$-satisfiable but there is a valid valuation θ of (T_{Q_1}, u) taking values from $\mathbb{D}$ such that $\theta(u) \notin Q_2(\theta(T_Q))$. By the characterization of $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$ given earlier, it then follows that $Q_1 \not\sqsubseteq_{\mathcal{A}} Q_2$. Conversely, if $Q_1 \not\sqsubseteq_{\mathcal{A}} Q_2$ then Q_1 must be $\mathcal{A}$-satisfiable and there exists a valid θ' of (T_{Q_1}, u) taking values from some domain $\mathbb{D}'$ such that $\theta'(u) \notin Q_2(\theta'(T_Q))$. A similar argument as given in the proof of Lemma 1 shows that θ' can be turned into a valid valuation θ of (T_{Q_1}, u) by taking values from $\mathbb{D}$ such that $\theta(u) \notin Q_2(\theta(T_Q))$. In other words, such a θ will be guessed by the algorithm and "yes" is returned.

Lower bound. We show that it is Π_2^p-hard to decide whether $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$, by reduction from the $\forall^*\exists^*$3CNF problem, which is known to be Π_2^p-complete [Sto76]. The $\forall^*\exists^*$3CNF problem is to decide, given a sentence $\varphi = \forall X_\psi \exists Y_\psi \, \psi(X_\psi, Y_\psi)$, whether φ is true. Here $X_\psi = \{x_1, \ldots, x_m\}$, $Y_\psi = \{y_1, \ldots, y_n\}$ and ψ is an instance of SAT defined on variables in $X_\psi \cup Y_\psi$.

Given an instance $\varphi = \forall X_\psi \exists Y_\psi \, \psi(X_\psi, Y_\psi)$ of the $\forall^*\exists^*$3CNF problem, we define a relational schema $\mathcal{R}$, an access schema $\mathcal{A}$ and two CQ Q_1 and Q_2 over $\mathcal{R}$, such that φ is true iff $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$. We define $\mathcal{R}$, $\mathcal{A}$ and Q as follows.

(a) The relational schema $\mathcal{R}$ consists of $m+6$ relation schemas: $S_i(X)$ for each $i \in [1, m]$, $R_{01}(X)$, $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$, $R_\neg(A, \bar{A})$, $R_s(A)$ and $R_b(A)$, where m is the number of variables in X_ψ. More specifically,

- unary relation $S_i(X)$ is to encode a truth assignment for variable x_i in X_ψ; these unary relations, when put together, encode a truth assignment of X_ψ;
- R_{01}, $R_\vee$, $R_\wedge$ and $R_\neg$ are the same as their counterparts given in the proof of Lemma 1; and

- two additional unary relation schemas $R_s(A)$ and $R_b(A)$. These will be used to indicate whether query Q is boundedly evaluable or not. Furthermore, an access constraint is imposed on $R_b(A)$ to control the cardinality of its instances, no constraints are defined over R_s.

(b) The access schema $\mathcal{A}$ consists of $m+5$ access constraints, as follows:

- a constraint $S_i(\emptyset \to X, 1)$ for each $i \in [1,m]$, to ensure that at any time, x_i takes a single truth value;
- constraints on $R_{01}, R_\vee, R_\wedge$ and $R_\neg$ to make sure that the instances of these relation schemas indeed encode Boolean values and operations: $R_{01}(\emptyset \to X, 2)$, $R_\vee(\emptyset \to (B, A_1, A_2), 4)$, $R_\wedge(\emptyset \to (B, A_1, A_2), 4)$, and $R_\neg(\emptyset \to (A, \bar{A}), 2)$; these are the same as their counterparts given in the proof of Lemma 1; and
- a constraint $R_b(\emptyset \to A, N)$, where N is a natural number, indicating that the cardinality of instances I_b of R_b is "bounded".

(c) Queries $Q_1(x)$ and $Q_2(x)$ in CQ are defined as

$$Q_1(x) = \exists u(Q_c \wedge Q_\varphi(u) \wedge R_s(x) \wedge R_b(x) \wedge u = 0), \quad Q_2(x) = \exists u(Q_c \wedge Q_\varphi(u) \wedge R_b(x) \wedge u = 1).$$

Query Q_c is the same as its counterpart given in the proof of Lemma 1. Query Q_φ encodes the given instance φ of the $\forall^*\exists^*$3CNF problem, and is defined as follows:

$$Q_\varphi(u) = \exists \bar{x} \exists \bar{y} \big(Q_{X_\psi}(\bar{x}) \wedge Q_{Y_\psi} \ \wedge\ Q_\psi(u, \bar{x}, \bar{y})\big),$$

where $Q_{X_\psi}(\bar{x}) = \bigwedge_{i=1}^{m} \big(S_i(x_i) \wedge R_{01}(x_i)\big)$ selects a valid truth assignment μ_X for variables X_ψ from the instances of S_i, and $Q_{Y_\psi} = \bigwedge_{i=1}^{n} R_{01}(y_i)$ selects a truth assignment μ_Y for Y_ψ from I_{01}. Query $Q_\psi(u, \bar{x}, \bar{y})$ encodes the truth value of the SAT instance $\psi(X_\psi, Y_\psi)$ for given truth assignments μ_X for X_ψ and μ_Y for Y_ψ, such that $u = 1$ if ψ is satisfied by μ_X and μ_Y, and $u = 0$ otherwise. Query Q_ψ can be expressed in CQ in the same way as its counterpart given in the proof of Lemma 1.

Note that Q_c and Q_φ are CQ, and hence so are Q_1 and Q_2. Furthermore, if Q_c is nonempty when posed on an instance D of $\mathcal{R}$ that satisfies $\mathcal{A}$, the instances of R_{01}, $R_\vee, R_\wedge$ and $R_\neg$ in D correctly encode the Boolean domain and operations $\vee, \wedge$ and $\neg$, respectively. Similarly, if Q_{X_ψ} is non-empty on D then the instances of S_i in D encode a valid truth assignment for X_ψ.

We next show that $\mathcal{R}$, $\mathcal{A}$ and Q are indeed a reduction, *i.e.,* φ is true iff $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$.

$\boxed{\Rightarrow}$ If φ is true, then for all truth assignments μ_X for X_ψ, there exists a truth assignment μ_Y for Y_ψ such that μ_X and μ_Y satisfy $\psi(X_\psi, Y_\psi)$. Hence in any instance D of $\mathcal{R}$ such that $D \models \mathcal{A}$, if Q_c and Q_{X_ψ} are nonempty, then for the truth assignment μ_X^D encoded in D, there exists μ_Y^D in $Q_{Y_\psi}(D)$ such that $Q_\psi(u, \mu_X^D, \mu_Y^D) \wedge u = 1$ is nonempty by the definition of Q_ψ. Therefore, $Q_2(D)$ is the instance I_b of R_b in D. Moreover, $Q_1(D)$ is the intersection of I_b and the instance I_s of R_s in D in this case. Hence $Q_1(D) \subseteq Q_2(D)$. If either Q_c or Q_{X_ψ} is empty on D, both $Q_1(D)$ and $Q_2(D)$ are empty, and hence again $Q_1(D) \subseteq Q_2(D)$. Note that when D ranges over all instances of $\mathcal{R}$ that satisfy $\mathcal{A}$, *all truth assignments* of X_ψ are checked. When D varies and $Q_{X_\psi}(D)$ is nonempty, only the truth assignments for X_ψ and the instances of R_b and R_s in D change, while the instances of R_{01}, $R_\vee$, $R_\wedge$ and $R_\neg$ remain the same as long as Q_c is nonempty.

$\boxed{\Leftarrow}$ Conversely, if φ is false, then there exists a truth assignment μ_X for X_ψ such that for all truth assignments μ_Y for Y_ψ, μ_X and μ_Y do not satisfy $\psi(X_\psi, Y_\psi)$. Hence by the definitions of $\mathcal{R}$, $\mathcal{A}$ and Q, there exists an instance D of $\mathcal{R}$ such that $D \models \mathcal{A}$, $Q_c(D)$ and $Q_{X_\psi}(D)$ are nonempty, $Q_\psi(u, \mu_X, \mu_Y^D) \wedge u = 1$ is empty for the truth assignment μ_X encoded in D and all truth assignment μ_Y^D returned by $Q_{Y_\psi}(D)$. By contrast, $Q_\psi(u, \mu_X, \mu_Y^D) \wedge u = 0$ is nonempty in D. In other words, $Q_1(D) \not\subseteq Q_2(D)$ in this case. Hence $Q_1 \not\sqsubseteq_{\mathcal{A}} Q_2$. $\square$

2.2.2 Bounded Evaluability of CQ

We next show that, as opposed to BEP(FO), which is undecidable, the BEP analysis is decidable for CQ. Below we first show the lower bound for BEP(CQ), and then prove the decidability.

Theorem 3: BEP(CQ) *is* EXPSPACE-*hard.* $\square$

Proof: We prove that BEP(CQ) is EXPSPACE-hard by reduction from the non-emptiness problem for parameterized regular expressions, denoted by NON-EMPTINESS$_\square$, which is known to be EXPSPACE-complete [BRL13]. A parameterized expression e is regular expression over $\Sigma \cup \Gamma$, where Σ is a finite set of symbols and Γ is a countably infinite set of variables disjoint from Σ. Of course, e uses only a finite number of variables from Γ. A valuation ν over e is a mapping from variables in e to Σ such that $\mu(e)$ is a regular expression over Σ. Let $\mathcal{L}(\nu(e))$ be the language accepted by $\nu(e)$ and define $\mathcal{L}_\square(e) = \bigcap\{L(\nu(e)) \mid \nu$ is a valuation for e over $\Sigma\}$, *i.e.*, $\mathcal{L}_\square(e)$ is the intersection of all languages of $\nu(e)$ for all valuations of e over Σ. The NON-EMPTINESS$_\Box$ problem is to decide whether $\mathcal{L}_\Box(e)$ is not empty, *i.e.,* whether there exists a word $w \in \Sigma^*$ such that $w \in \mathcal{L}_\Box(e)$.

A parameterized regular expression e over $\Sigma \cup \Gamma$ can be equivalently represented by a a nondeterministic finite automaton $A_e = (s_0, S, \Sigma \cup \Gamma, \delta, s_f)$, where s_0 is the initial state, S is the set of states, $\Sigma \cup \Gamma$ is the input alphabet, $\delta : S \times \Sigma \cup \Gamma \rightarrow S$ is the state transition function and s_f is the final state. The translation from e to A_e takes polynomial time. For a valuation ν of e we denote by $\nu(A_e)$ the automaton obtained by replacing the variables in Γ that occur in δ by their corresponding symbol in Σ, as determined by ν. Clearly, for any valuation ν, $\mathcal{L}(\nu(A_e)) = \mathcal{L}(\nu(e))$. That is, $\nu(A_e)$ and $\nu(e)$ accept the same regular language. In what follows, we assume that Σ consists of K symbols $a_1, \ldots, a_K$.

Given a parameterized regular expression e over $\Sigma \cup \Gamma$, we define a relational schema $\mathcal{R}$, an access schema $\mathcal{A}$ over $\mathcal{R}$ and a boolean conjunctive query Q over $\mathcal{R}$, and then show that $\mathcal{L}_\Box(e)$ is not empty if and only if Q has a boundedly evaluable plan.

We give the reduction as follows.

(1) The relational schema consists of three relation schemas as follows: (a) $T(Z_1, S_1, V, S_2)$, which is to encode the state transition function δ of A_e; (b) $S(Z_2, W)$, to represent constants in $\Sigma \cup \Gamma$; and (c) $F(S, Z_3)$, which encodes the final state s_f of A_e.

(2) The access schema consists of the following four constraints: (a) $T(Z_1 \rightarrow (S_1, V, S_2), N)$ and $T((S_1, V) \rightarrow S_2, N)$, where N is an integer larger than the number of transitions in A_e; (b) $S(Z_2 \rightarrow W, K)$ where K is the size of Σ. This constraint is to ensure that all valuations of e are considered, as will be explained in more detail below; and (c) $F(S \rightarrow Z_3, 1)$, which is used for the unique final state s_f.

(3) The Boolean query $Q()$ is defined as follows:

$$Q() = \exists z \bar{x} \bar{y} \big(Q_{\mathsf{eval}}(z, \bar{x}_e) \wedge Q_f(z, \bar{x}_f) \wedge Q_\delta(z, \bar{x}, \bar{y}) \big),$$

Here $\bar{x}$ are variables for input alphabet $\Sigma \cup \Gamma$ including those in $\bar{x}_e$ encoding variables in e, and $\bar{y}$ are variables for states of A_e, while z is an extra variable not for encoding A_e and will be explained shortly.

- ◦ $Q_{\mathsf{eval}}(z, \bar{x}_e) = \bigwedge_{i=1}^{K} S(z, b_i) \wedge \bigwedge_{j=1}^{M} S(z, x_e^j)$, where variables $\bar{x}_e = x_e^1, \ldots, x_e^M$ encode all valuations of variables in e; indeed, when Q_{eval} is not empty on a database D where $D \models \mathcal{A}$, the instance I_S of S must encode the set Σ, based on the definition of the subquery $\bigwedge_{i=1}^{K} S(z, b_i)$ and constraint $S(Z \to X, K)$, and then Q_{eval} can return all valuations of variables in e;

- $Q_f(x_f, z) = F(x_f, z)$, where variable x_f is to encode the final state s_f of A_e; and
- $Q_\delta(z, \bar{x}, \bar{y}) = \bigwedge_{s,s' \in S, b \in \Sigma, \delta(s,b)=s'} T(z, y_s, b, y_{s'}) \wedge \bigwedge_{s,s' \in S, b \in \Gamma, \delta(s,b)=s'} T(z, y_s, x_b, y_{s'}) \wedge y_{s_0} = s_0$ is to encode the state transition function δ in A, where variables $\bar{x}$ represent all variables in e, $\bar{y}$ encodes all states of A_e, and variable $y_{s_0} \in \bar{y}$ represents the initial state s_0; here $y_{s_0} = s_0 \notin \Sigma$ guarantees that when simulating A_e, Q can start with the initial state s_0.

We show that $\mathcal{L}_\Box(e)$ is not empty if and only if Q has a bounded query plan.

$\boxed{\Rightarrow}$ Assume that $\mathcal{L}_\Box(e)$ is not empty. Then there exists a word $\bar{w} = b'_1 \dots b'_p \in \mathcal{L}_\Box(e)$. That is, for any valuation ν of e over Σ, $\bar{w}$ is in $L(\nu(e))$. Define query Q' as follows:

$$Q' = \exists z \bar{x} \bar{y} \bar{w} \big(Q_{\mathsf{eval}}(z, \bar{x}_e) \wedge Q_f(x_f, z) \wedge Q_\delta(z, \bar{x}, \bar{y}) \wedge Q_w(z, \bar{z}, \bar{w}) \big),$$

where $Q_w(\bar{z}, \bar{w}) = T(z_1, s_0, b'_1, w_1) \wedge T(z_2, w_1, b'_2, w_2) \wedge \dots \wedge T(z_p, w_{p-1}, b'_p, w_p) F(w_p, z)$. Here $\bar{w} = w_1, \dots, w_p$ and $\bar{z} = z_1, \dots, z_p$ are $2p$ pairwise distinct variables not in Q. We can readily verify that Q' is covered by $\mathcal{A}$. Moreover, we can show that Q' is $\mathcal{A}$-equivalent to Q. Indeed, $Q'() \sqsubseteq_{\mathcal{A}} Q()$ since $Q'() \subseteq Q()$. It remains to show that $Q() \sqsubseteq_{\mathcal{A}} Q'()$.

Define $Q_\nu() = Q() \wedge \bigwedge_{i=1}^{M} x_e^i = b'_i$, where $x_e^i (i \in [1, M])$ encode all variables in e as discuss above, and all $b'_i (i \in [1, M])$ are constants in Σ. Intuitively, $\bigwedge_{i=1}^{M} x_e^i = b'_i$ encodes a evaluation ν of e over Σ. Obviously, $Q \equiv_{\mathcal{A}} \bigcup \{Q_\nu() \mid \nu \text{ is any valuation of } e \text{ over } \Sigma\}$ and $Q_\nu() \subseteq Q$ for any evaluation ν of e over Σ.

Consider any Q_ν and its tableau T_{Q_ν}. Since $\bar{w}$ can be accepted by $\mathcal{L}(\nu(e))$, there exists a run $s_0 \xrightarrow{b'_1} s_1 \dots \xrightarrow{b'_l} s_l$ of $\nu(A_e)$ over $\bar{w}$ such that $s_l = s_f$. By mapping $Q_w(z, \overline{w})$ to the tuples corresponding to this run, we can construct a homomorphism h_1 from $Q_w(z, \bar{w})$ to T_{Q_ν}, where $h_1(z) = z$. From the definition of Q_ν, it is easy to see that there exists a homomorphism h_2 from Q to Q_ν such that $h_2(z) = z$. By combining h_1 and h_2, we can obtain a homomorphism from Q' to Q_ν. That is $Q_\nu \subseteq Q'$. Because $Q \equiv_{\mathcal{A}} \bigcup \{Q_\nu() \mid \nu \text{ is any valuation of } e \text{ over } \Sigma\}$, we can conclude that $Q \subseteq_{\mathcal{A}} Q_1$.

$\boxed{\Leftarrow}$ Suppose that Q has a boundedly evaluable query plan ξ. We assume *w.l.o.g.* that ξ is minimal, *i.e.,* no operation can be removed from ξ. Then we have the following properties for ξ:

(a) For each $T \in \xi$, let $\xi_1 = T_1, T_2, \dots, T$ be a prefix of ξ ending with T. Then if there exists a database D such that $\xi_1(D) = \emptyset$, then $\xi(D) = \emptyset$.

(b) There exists a sequence ξ' of fetching operations in ξ of the form $\mathsf{fetch}(\{c,b'_1\}, T, y_1)$, $\mathsf{fetch}(\{y_1, b'_2\}, T, y_2)$, $\ldots$, $\mathsf{fetch}(\{y_{l-1},b'_l\}, T, y_l)$, $\mathsf{fetch}(\{y_l\}, F, z)$ corresponding to relation atoms $T(Z_1,c,b'_1,y_1),T(Z_2,y_1,b'_2,y_2),\ldots,T(Z_l, y_{l-1},b'_l,y_l),F(y_l,z)$ in Q for some $b'_1, \ldots, b'_l$ in Σ, such that ξ' encodes a word $\bar{w} = b'_1 b'_2 \ldots b'_l \in (\Sigma)^*$. This is because, by the construction of access constraints, before fetching z any boundedly evaluable query plan can only fetch values for variables via index in $T((X_1,B) \to X_2,N)$, starting with constants in Q.

Using these properties, we can show $\mathcal{L}_\Box(e) \neq \emptyset$ by contradiction as follows. Suppose $\mathcal{L}_\Box(e) = \emptyset$. Then there exists a valuation ν for e such that $\bar{w} \notin \mathcal{L}(\nu(e))$. Consider Q_ν of ν (see the definition of Q_ν above). Let $Q_{\xi'}(z) = T(Z_1,c,b'_1,y_1),T(Z_2,y_1,b'_2,y_2), \ldots,T(Z_l,y_{l-1},b'_l,y_l),F(y_l,z)$. Since $\bar{w} \notin \mathcal{L}(\nu(A_e))$, there does not exist a run $s_0 \xrightarrow{b'_1} s_1 \ldots \xrightarrow{b'_l} s_l$ of $\nu(A_e)$ over $\bar{w}$ such that $s_l = s_f$. Then there does not exist a valuation of $Q_{\xi'}$ in T_{Q_ν} (that is treated as a database). That is, $\xi'(T_{Q_\nu}) = Q_{\xi'}(T_{Q_\nu}) = \emptyset$. Based on properties (a) and (b) above, we know that $\xi(T_{Q_\nu}) = \emptyset$. On the other hand, since Q_ν is of the form $Q \wedge x_e^0 = b'_1 \wedge \ldots \wedge x_e^M = b'_M$, we have $Q(T_{Q_\nu}) \neq \emptyset$. Then we have $Q(T_{Q_\nu}) \neq \xi(T_{Q_\nu})$, which is a contradiction. Therefore $\mathcal{L}_\Box(e) \neq \emptyset$. $\Box$

Despite the high lower bound, $\mathsf{BEP(CQ)}$ is decidable.

Theorem 4: $\mathsf{BEP(CQ)}$ *is in* $\mathsf{2EXPSPACE}$. $\Box$

Proof: Below we develop a $\mathsf{2EXPSPACE}$ algorithm for $\mathsf{BEP(CQ)}$. The algorithm is based on a characterization of boundedly evaluable queries. Below we first give necessary notions, based on which we then give the characterization (Lemma 6 below). We then give the algorithm, and verify its correctness by proving the characterization.

(1) Notions. We use the following notions.

Element query. A CQ query Q_e is called an *element query* of a CQ query Q under an access schema $\mathcal{A}$ if $Q_e = Q \wedge \phi$, where ϕ is a conjunction of equality atoms between variables in Q, such that the tabeau representation T_e of Q_e satisfies $\mathcal{A}$, *i.e.*, $T_e \models \mathcal{A}$ when T_e is viewed as an instance, by treating variables as constants. We say that Q_e *satisfies* $\mathcal{A}$ if its tableau satisfies $\mathcal{A}$ as above.

Observe the following.

- A CQ Q has at most exponentially many element queries, since there are only $O(2^{|Q|})$ possible ϕ.
- We have that $Q \equiv_\mathcal{A} Q_{e_1} \cup \cdots \cup Q_{e_n}$, where Q_{e_i}'s are all element queries of Q under $\mathcal{A}$ for $i \in [1,n]$.

- An element query may not be satisfiable. For example, consider $\mathcal{R}$ consisting of a single relation $R(X,Y)$, $Q(x) = R(k,x_1) \wedge R(k,x_2) \wedge R(k,x_3) \wedge R(x_3,x) \wedge (x_1 = 1) \wedge (x_2 = 2)$, and $\mathcal{A} = \{R(X \to Y,2)\}$. Then Q has seven possible element queries: $Q_1(x) = Q(x) \wedge (x_1 = x_2)$, $Q_2(x) = Q(x) \wedge (x_2 = x_3)$, $Q_3(x) = Q(x) \wedge (x_1 = x_3), \ldots, Q_7(x) = Q(x) \wedge ((x_1 = x_3) \wedge (x_1 = x_2) \wedge (x_2 = x_3))$. Among these, Q_1 and Q_7 are not satisfiable. However, since the tableau of an element query Q_e satisfies $\mathcal{A}$, it can be checked in PTIME whether Q_e is satisfiable. Hence in the sequel we consider *w.l.o.g.* only satisfiable element queries.

Intuitively, element queries make the analysis of bounded output easier. When the tableau of Q does not satisfy $\mathcal{A}$, it is nontrivial to check whether variables in Q have a bound on their valuations. Considering $Q(x)$ above as an example, we do not know whether there exists a bound on the valuation of x_3. In contrast, when considering element queries $Q_2(x)$ and $Q_3(x)$, we can easily see the bounds on valuations of x_3.

Element expansion. Let S be a subset of all the element queries of CQ query Q under access schema $\mathcal{A}$ such that: (i) each Q_e in S is satisfies $\mathcal{A}$ and is satisfiable; (ii) for any element query Q_e of Q under $\mathcal{A}$, if $Q_e \notin S$, then there exists $Q'_e \in S$ such that $Q_e \sqsubseteq Q'_e$; if $Q_e \in S$, then for any other $Q'_e \in S$, $Q_e \not\sqsubseteq Q'_e$. Here we use $Q_1 \sqsubseteq Q_2$ to denote $Q_1(D) \subseteq Q_2(D)$ for any D, *i.e.,* the conventional query containment. We refer to $\bigcup_{Q_e \in S} Q_e$ as an *element expansion* of Q under $\mathcal{A}$. Intuitively, the element expansion is an expression of Q in terms of its "non-redundant" element queries under $\mathcal{A}$.

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

We next inductively define the set $\mathsf{cov}(Q,\mathcal{A})$ of *variables covered by* $\mathcal{A}$, starting from $\mathsf{cov}_0(Q,\mathcal{A}) = \emptyset$. When $i > 0$, we say that an access constraint $\varphi = R(X \to Y,N)$ is *applicable* to an atom $R(\bar{x},\bar{y},\bar{z})$ in Q if

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

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

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

We define $\mathsf{cov}(Q, \mathcal{A}) = \mathsf{cov}_k(Q, \mathcal{A})$ when $\mathsf{cov}_k(Q, \mathcal{A}) = \mathsf{cov}_{k+1}(Q, \mathcal{A})$, *i.e.,* as "the fixpoint".

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

Lemma 5: *For any* CQ *query Q and access schema $\mathcal{A}$ over relational schema $\mathcal{R}$,* $\mathsf{cov}(Q, \mathcal{A})$ *is uniquely determined and can be computed in* PTIME *in $|Q|$, $|\mathcal{R}|$ and* $|\mathcal{A}|$. □

<u>*Covered queries*</u>. A CQ $Q(\bar{x})$ is *covered by $\mathcal{A}$* if

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

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

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

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

Example 3: Query Q_3 of Example 2 is covered by $\mathcal{A}_3$. Indeed, (a) $\mathsf{cov}(Q_3, \mathcal{A}_3) = \{x, y, z_3, x_1, x_2\}$, including free variables x and y; (b) while variables z_1 and z_2 are uncovered, they satisfy condition (b) above; and (c) relation atoms $R(x_1, x_2, x)$ and $R(x, y, z_3)$ are indexed by φ_5, and $R(z_1, z_2, y)$ is indexed by φ_4.

In contrast, Q_1 of Example 2 is not covered by $\mathcal{A}_1$: Q_1 does not satisfy condition (c), since $R(x_1, x, x_2, y)$ is not indexed by any constraint in $\mathcal{A}_1$.

As another example, query Q_0 of Example 1 in Chapter 1 is covered by $\mathcal{A}_0$

consisting of ψ_1–ψ_4. Indeed, its free variable x_a is covered, non-covered variables cid and class occur only once in Q_0, and all its relation atoms are indexed: Accident by ψ_3, Casualty by ψ_2 and Vehicle by ψ_4. □

Note that, by Lemma 5, it is in PTIME to decide whether a CQ query Q is covered by an access schema $\mathcal{A}$.

(2) Characterization. We are now ready to present the characterization of boundedly evaluable CQ queries. It is given in Lemma 6 as follows.

Lemma 6: *Under an access schema $\mathcal{A}$, a* CQ *query Q is boundedly evaluable under $\mathcal{A}$ if and only if there exists a* CQ *query Q' such that*

(1) Q' is covered by $\mathcal{A}$; and

(2) $|Q'| \leqslant |Q|^{(1+(|Q|+1)\cdot 2^{|Q|})}$. □

(3) Algorithm. Based on the characterization in Lemma 6, we have the following algorithms for BEP(CQ).

1. Guess a CQ query Q' of size no larger than $|Q|^{(1+(1+|Q|)\cdot 2^{|Q|})}$.
2. Check whether $Q' \equiv_{\mathcal{A}} Q$ and Q' is covered by $\mathcal{A}$; return "yes" if so.

The correctness of the algorithm is guaranteed by Lemma 6(1). By Lemma 6(2), $O(|Q'|) = 2^{2^{O(|Q|)}}$. Further by Lemma 2 and Lemma 5, we know that the algorithm is in 2NEXPSPACE = 2EXPSPACE.

To complete the upper bound proof, below we prove Lemma 5 and Lemma 6.

Proof of Lemma 5. Let $\mathsf{cov}_{i-1}(Q,\mathcal{A})$ be computed so far and consider $\mathsf{cov}_i(Q,\mathcal{A})$ and $\mathsf{cov}'_i(Q,\mathcal{A})$ obtained by applying two distinct access constraints $\varphi_1 = R(X_1 \to (Y_1,N_1))$ and $\varphi_2 = S(X_2 \to (Y_2,N_2))$ in $\mathcal{A}$. Note that if φ_1 was applicable *w.r.t.* $\mathsf{cov}_{i-1}(Q,\mathcal{A})$ it still applicable *w.r.t.* $\mathsf{cov}'_i(Q,\mathcal{A})$ (unless all variables in the respective relation atom are already covered, in which case the application of φ_1 has no effect); similarly for φ_2 and $\mathsf{cov}_i(Q,\mathcal{A})$. It is readily verified that the application of φ_1 on $\mathsf{cov}'_i(Q,\mathcal{A})$ and φ_2 on $\mathsf{cov}_i(Q,\mathcal{A})$ results in the same set. This suffices to show that $\mathsf{cov}(Q,A)$ is independent of the order in which the constraints are applied.

This also implies that the computation of $\mathsf{cov}(Q,\mathcal{A})$ can be done by iteratively applying all applicable constraints simultaneously, until no applicable constraints are left. Observe that in each step, at least one variable in Q became covered. Hence, the number of iterations is at most the number of variables, i.e., $O(|Q|\cdot|\mathcal{R}|)$. Since in each step we check $|\mathcal{A}|$ constraints, the overall complexity is $O(|Q||\mathcal{R}||\mathcal{A}|)$. □

Proof of Lemma 6. This is more involved. It is based on the following two lemmas.

Lemma 7: *Let $Q^e_1 \cup \ldots \cup Q^e_m$ and $Q'^e_1 \cup \ldots \cup Q'^e_n$ are element expansions of* CQ *queries Q and Q' under access schema $\mathcal{A}$, respectively, then*

(1) $Q \sqsubseteq_{\mathcal{A}} Q'$ if and only if $Q^e_1 \cup \ldots \cup Q^e_m \sqsubseteq Q'^e_1 \cup \ldots \cup Q'^e_n$; and

(2) $Q \equiv_{\mathcal{A}} Q'$ if and only if $Q^e_1 \cup \ldots \cup Q^e_m \equiv Q'^e_1 \cup \ldots \cup Q'^e_n$ and $m = n$.

Here $Q \sqsubseteq Q'$ (resp. $Q \equiv Q'$) if for any D, $Q(D) \subseteq Q(D')$ (resp. $Q(D) = Q(D')$), i.e., *conventional query containment (resp. equivalence).* □

Lemma 8: *Under access schema $\mathcal{A}$, a* CQ *query Q is boundedly evaluable under $\mathcal{A}$ if and only if there exists a query Q' covered by $\mathcal{A}$ such that $Q \equiv_{\mathcal{A}} Q'$;* □

Below we first use Lemma 7 and Lemma 8 to prove Lemma 6. We then prove Lemma 7 and Lemma 8 in the end.

By Lemma 8, we know that there exists a CQ query Q^c that is covered by $\mathcal{A}$ and $Q^c \equiv_{\mathcal{A}} Q$ if a CQ query Q is boundedly evaluable under $\mathcal{A}$. Below we prove Lemma 6 in two steps.

(a) We first construct another query Q^{c_0} from Q^c *w.r.t.* Q such that Q^{c_0} is also covered by $\mathcal{A}$, such that $Q^{c_0} \equiv_{\mathcal{A}} Q$; and

(b) We then derive query $Q^{c_0}_m$ from Q^{c_0} such that (i) $Q^{c_0}_m \equiv_{\mathcal{A}} Q^{c_0}$, (ii) $Q^{c_0}_m$ is also covered by $\mathcal{A}$, and (iii) the size $|Q^{c_0}_m|$ of $Q^{c_0}_m$ is bounded by $|Q|^{(1+(|Q|+1)\cdot 2^{|Q|})}$.

(a) We construct Q^{c_0} from Q^c via homomorphisms from Q^c to element queries of Q. Let $Q^c_1 \cup \ldots \cup Q^c_m$ and $Q_1 \cup \ldots \cup Q_n$ be the element expansions of Q^c and Q under $\mathcal{A}$, respectively. Then $Q^c_1 \cup \ldots \cup Q^c_m \equiv_{\mathcal{A}} Q_1 \cup \ldots \cup Q_n$. By Lemma 7, $Q^c_1 \cup \ldots \cup Q^c_m \equiv Q_1 \cup \ldots \cup Q_n$ and $m = n$ (below we use n for both Q^c and Q). Therefore, for each Q^c_i of Q^c, there exists a $Q_j (j \in [1,n])$ of Q such that $Q^c \equiv Q_j$ (cf. [SY80]). For convenience, assume that by ordering, for each $i \in [1,n]$, $Q_i \equiv Q^c_i$. Thus there exists a homomorphism from Q^c_i to Q_i for each $i \in [1,n]$. Since there exists a homomorphism from Q^c to each of its element queries Q^c_i, there must exist a homomorphism h_i from Q^c to each element query Q_i of Q ($i \in [1,n]$). Now consider a renaming function ρ that renames variables in Q^c *w.r.t.* $h_i (i \in [1,n])$ as follows:

- for each variable x in Q^c, if x occurrs twice or is a free variable, then $\rho(x) = (h_1(x), \ldots, h_n(x))$;
- for any other variable y (*i.e.,* those existentially quantified variables that occur

only once in Q), $\rho(y) = y$.

Let Q^{c_0} be the query derived from Q^c via ρ. By the definition of ρ, for each element query Q_i of Q, there exists a homomorphism h'_i from Q^{c_0} to Q_i: for each variable of form $(x_1,\ldots,x_n)$ in Q^{c_0}, $h'_i(x_1,\ldots,x_n) = x_i$; and for any other variable x in Q^{c_0} (*i.e.,* not renamed variables), $h'_i(x) = h_i(x)$. Since h_i is a homomorphism from Q^c to Q_i, h'_i is also a homomorphism from Q^{c_0} to Q_i. Thus $Q \equiv_{\mathcal{A}} Q_1 \cup \cdots Q_n \sqsubseteq Q^{c_0}$. Furthermore, since ρ is a function, it is a homomorphism from Q^c to Q^{c_0} already. Thus $Q^{c_0} \sqsubseteq Q^c \equiv_{\mathcal{A}} Q$. Therefore, $Q^{c_0} \equiv_{\mathcal{A}} Q$.

Moreover, Q^{c_0} is also covered by $\mathcal{A}$ when Q^c is. Indeed, observe that all existentially quantified variables that occur only once are kept unchanged via the ρ. Since all constants in Q are mapped to constants in Q' via ρ, and by the definition of covered queries, $\mathsf{cov}(Q,\mathcal{A})$ deduces new covered variables from constants and existing covered variables only. Therefore, if $x \in \mathsf{cov}(Q^c,\mathcal{A})$, then $\rho(x) \in \mathsf{cov}(Q^{c_0},\mathcal{A})$. Thus Q^{c_0} is also covered by $\mathcal{A}$ when Q^c is.

(b) We next derive $Q^{c_0}_m$ from Q^{c_0} by applying the following rule:

- ○ for each relation atom $R(\bar{x},\bar{y})$ in Q^{c_0}, if (i) $\bar{y}$ consists of variables in Q^c that are not renamed in Q (*i.e.,* existentially quantified variables in Q^{c_0} that occur only once); and (ii) there exists another relation atom $R(\bar{x},\bar{y}')$ in Q^{c_0} such that $\bar{y}'$ also consists of variables that are not renamed, then remove $R(\bar{x},\bar{y})$ from Q^{c_0}.

Obviously, $Q^{c_0}_m \equiv Q^{c_0}$ since each time we remove $R(\bar{x},\bar{y})$ that is redundant due to the existence of $R(\bar{x},\bar{y}')$. In addition, $Q^{c_0}_m$ is also covered by $\mathcal{A}$ since Q^{c_0} is. Indeed, observe that the application of the above rule has no impact on constants and variables that occur more than once or are free. Therefore, $x \in \mathsf{cov}(Q^{c_0},\mathcal{A})$, then $x \in \mathsf{cov}(Q^{c_0}_m,\mathcal{A})$ as well. Thus $Q^{c_0}_m$ is also covered by $\mathcal{A}$ when Q^{c_0} is.

We next study the size of $Q^{c_0}_m$. First observe that, by the definition of ρ, there are at most $N = |Q|^n$ distinct variables in $Q^{c_0}_m$ that are not trivial, and n is no larger than $C^0_{|Q|} + \cdots + C^{|Q|}_{|Q|} = 2^{|Q|}$ by the definition element expansion. Thus $N = |Q|^{2^{|Q|}}$. In addition, by the construction of $Q^{c_0}_m$, we know that there are at most $(N + N^2 + \cdots N^{|\mathcal{R}|}) = \frac{N^{|\mathcal{R}|+1}-N}{N-1}$ tuples in the tableau of $Q^{c_0}_m$, each is of size bounded by $|\mathcal{R}|$ which is no larger than $|Q|$. Therefore, $|Q^{c_0}_m| \leqslant \frac{N^{|\mathcal{R}|+1}-N}{N-1} \times |\mathcal{R}| < |Q| \cdot N^{|Q|+1} = |Q|^{(1+(|Q|+1)\cdot 2^{|Q|})}$. This completes the proof of Lemma 6 □

We next prove Lemma 7 and Lemma 8.

Proof of Lemma 7. To see Lemma 7 is correct, we only need to show Lemma 7(1) is correct, since $m = n$ if $Q^e_1 \cup \cdots Q^e_m$ and $Q'^e_1 \cup \cdots Q'^e_n$ are element expansions of Q (*i.e.,* non-redundant) and are equivalent (cf. [SY80]). Since $Q \sqsubseteq_{\mathcal{A}} Q'$ if $Q^e_1 \cup \cdots \cup Q^e_m \sqsubseteq Q'^e_1 \cup \cdots \cup Q'^e_n$ by the definition of $\mathcal{A}$-containment, we only need to show the other direction, *i.e.,* $Q^e_1 \cup \cdots \cup Q^e_m \sqsubseteq Q'^e_1 \cup \cdots \cup Q'^e_n$ if $Q \sqsubseteq_{\mathcal{A}} Q'$. We show this by verifying that, $Q^e_i \sqsubseteq Q'$ for each $i \in [1,m]$ when $Q \sqsubseteq_{\mathcal{A}} Q'$. That is to show, for any D, $Q^e_i(D) \subseteq Q'(D)$, *i.e.,* for any valuation θ of Q^e_i on D, $\theta(u_i) \in Q'(\theta(T_i))$, where (T_i, u_i) is the tableau of Q^e_i. Since $Q^e_i \sqsubseteq_{\mathcal{A}} Q'$, and $\theta(T_i) \models \mathcal{A}$ as Q^e_i is an element query, we know that $\theta(u_i) \in Q'(\theta(T_i))$ by Lemma 2. Thus $Q^e_i \sqsubseteq Q'$ by the monotonicity of Q. Therefore, when $Q \sqsubseteq_{\mathcal{A}} Q'$, $Q^e_1 \cup \ldots \cup Q^e_m \sqsubseteq Q'^e_1 \cup \ldots \cup Q'^e_n$. Thus Lemma 7 (1) is verified. □

Proof of Lemma 8. The proof is based on the following two lemmas, which will be verified shortly.

Lemma 9: *Every query plan ξ corresponds to covered* CQ Q_ξ *such that for any $D \models \mathcal{A}$, $\xi(D) = Q_\xi(D)$.* □

Lemma 10: *Let Q be a covered* CQ. *Then Q is boundedly evaluable.* □

Assuming these lemmas we have the following. Suppose first that Q is boundedly evaluable under $\mathcal{A}$ and let $\xi(Q, \mathcal{R})$ be a query plan for Q that is boundedly evaluable under $\mathcal{A}$. We have that for any $D \models \mathcal{A}$, $Q(D) = \xi(Q, \mathcal{R})(D)$. Lemma 9 tells us that there is covered CQ Q_ξ such that for any $D \models \mathcal{A}$, $Q_\xi(D) = \xi(Q, \mathcal{R})(D)$. That is, Q is indeed $\mathcal{A}$-equivalent to a covered CQ. Conversely, assume that Q is $\mathcal{A}$-equivalent to a covered CQ Q'. Lemma 10 tells us that Q' is boundedly evaluable. Let $\xi(Q', \mathcal{R})$ be a boundedly evaluable query plan under $\mathcal{A}$ for Q'. Then for any $D \models \mathcal{A}$, $Q(D) = Q'(D) = \xi(Q', \mathcal{R})(D)$. Hence, $\xi(Q', \mathcal{R})$ is also a boundedly evaluable query plan under $\mathcal{A}$ for Q and thus Q is boundedly evaluable under $\mathcal{A}$. □

It remains to verify Lemmas 9 and 10.

Proof of Lemma 9. Let ξ be a boundedly evaluable query plan under an access schema $\mathcal{A}$. We show the lemma in three steps: first we define the query Q_ξ associated with ξ, then show that $\xi \equiv_{\mathcal{A}} Q_\xi$, and finally verify that Q_ξ is covered by $\mathcal{A}$.

(1) Construction of Q_ξ. Since we only need Q_ξ for boundedly evaluable query plans, we only give the construction of Q_ξ of such plans. The query Q_ξ is inductively defined based on the structure of the query plan ξ. Let $\mathcal{R}$ be a relational schema. A query plan ξ under access schema $\mathcal{A}$ is a sequence

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

such that for all $i \in [1,n]$, δ_i is one of the following:

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

The result $\xi(D)$ of applying $\xi(\mathcal{R})$ to D is T_n. Given $\xi(\mathcal{R})$ we define Q_ξ as follows. In Q_ξ, we distinguish between three kinds of variables, x, $\hat{x}$ and $\tilde{x}$ indicating free covered, existentially quantified covered, and existentially uncovered variables, respectively. Vectors of such variables are denoted by $\bar{x}$, $\hat{\bar{x}}$ and $\tilde{\bar{x}}$, respectively.

1. If $\delta_i = \{a\}$, then we define $Q_i(x) := x = a$, where x is brand new variable.
2. If $\delta_i = \mathsf{fetch}(X \in T_j, R, Y)$, $j < i$, we define $Q_i(\bar{x}, \bar{y}, \hat{\bar{y}}, \tilde{\bar{z}}, \tilde{\bar{z}}') := Q_j(\bar{x}, \hat{\bar{y}}, \tilde{\bar{z}}) \wedge R(\bar{x}, \bar{y}, \tilde{\bar{z}}')$, where $\bar{y}$ and $\tilde{\bar{z}}'$ consist of brand new distinct variables, and $\bar{x}$ and $\bar{y}$ correspond to the attributes X and Y in R, respectively.
3. If $\delta_i = \rho_{A/B}(T_j)$, we define $Q_i(\bar{x}', \hat{\bar{y}}, \tilde{\bar{z}}) := Q_j(\bar{x}, \hat{\bar{y}}, \tilde{\bar{z}})$, where $\bar{x}'$ is obtained from $\bar{x}$ by switching the variables corresponding to attributes A and B.
4. If $\delta_i := \sigma_{A=B}(T_j)$ we define $Q_i(\bar{x}', \hat{\bar{y}}, \tilde{\bar{z}}) := Q_j(\bar{x}, \hat{\bar{y}}, \tilde{\bar{z}})$, where $\bar{x}'$ is obtained from $\bar{x}$ by unifying the variables corresponding to attributes A and B.
5. If $\delta_i := T_j \times T_k$ we define $Q_i(\bar{x}, \bar{x}', \hat{\bar{y}}, \hat{\bar{y}}', \tilde{\bar{z}}, \tilde{\bar{z}}') := Q_1(\bar{x}, \hat{\bar{y}}, \tilde{\bar{z}}') \wedge Q_2(\bar{x}', \hat{\bar{y}}', \tilde{\bar{z}}')$, where Q_1 and Q_2 have no variables in common.
6. Finally, if $\delta_i := \pi_U(T_j)$ we define $Q_i(\bar{x}', \hat{\bar{y}}', \tilde{\bar{z}}) := Q_j(\bar{x}, \hat{\bar{y}}, \tilde{\bar{z}})$, where $\bar{x}'$ is obtained by removing all unprojected variables from $\bar{x}$, which are added to $\hat{\bar{y}}$ to make up $\hat{\bar{y}}'$.

Let $Q_n(\bar{x}, \hat{\bar{y}}, \tilde{\bar{z}})$ be the query corresponding to T_n, the final step in the query plan. We then define the CQ query associated with ξ as

$$Q_\xi(\bar{x}) := \exists \hat{\bar{y}} \exists \tilde{\bar{z}} Q'_\xi(\bar{x}, \hat{\bar{y}}, \tilde{\bar{z}}).$$

In other words, we project away all covered variables marked as being existentially quantified and all uncovered variables appearing in Q_n.

(2) ξ is $\mathcal{A}$-equivalent to Q_ξ. We verify that for any instance D that satisfies $\mathcal{A}$, $\xi(D) = Q_x(D)$. We show this by induction on the *depth* of query plans. The depth of a query plan is defined as follows. Let ξ be a query plan of the form $T_1 = \delta_1, \ldots, T_n = \delta_n$, we say that T_i calls T_j for $j < i$ if T_j appears in the definition of T_i. We can stratify the

query plan as follows: ξ_0 consists of all T_i in ξ that do not call any other T_j. We call these the assignments in ξ of level 0. Then, for $d > 0$, we define ξ_d as the set consisting of all T_i in ξ that call a $T_j \in \xi_{d-1}(\mathcal{R})$. We call these the assignments in ξ of level d. A query plan ξ has depth d if $\xi = (\xi_0, \cdots, \xi_d)$ and $\xi_d \neq \emptyset$. We further assume that a query plan is connected, *i.e.,* every T_j in ξ_p is called by a T_i in ξ_q for $p < q$. Clearly, we may assume connected query plans.

Our induction hypothesis is that for any $D \models \mathcal{A}$, $\xi(D) = Q_\xi(D)$ for any query plan of depth at most d.

For the base case, when ξ has depth 0, we have that ξ consists of a single statement $T := \delta = \{a\}$ for some constant a. Clearly, $Q_\xi(x) = (x = a)$ is equivalent to ξ.

Suppose that the induction hypothesis holds for query plans of depth $< d$. Let ξ be a query plan of depth d. A straightforward case analysis shows that $\xi(D) = Q_\xi(D)$. Indeed, Q_ξ is the standard translation of relational algebra expressions into CQ, except for the case when $T_i = \mathsf{fetch}(X \in T_j, R, Y)$. However, for such statements the induction hypothesis immediately follows from the definition of fetch operators, *i.e.,* for $D \models \mathcal{A}$, the fetch operators retrieves all necessary tuples to evaluate Q_ξ.

(3) Q_ξ is covered by $\mathcal{A}$. As indicated by the kind of variables (free covered, existentially quantified covered or uncovered variables) shown in the construction of Q_ξ, we observe that when a new relation atom is introduced in Q_ξ by the fetch operations in ξ, all free variables and variables corresponding to constants are covered, and the uncovered variables are new existentially quantified distinct variables. Furthermore, when sub-queries are combined, the covered variables and distinctness of uncovered variables of the sub-queries are preserved in the combined query. In other words, being covered by $\mathcal{A}$ is preserved during the construction of Q_ξ starting from trivially covered constant variables $x = a$. □

Proof of Lemma 10. Let $Q(\bar{x}) = \exists \bar{y}\, P(\bar{x}, \bar{y})$ be a CQ over relational schema $\mathcal{R}$. Let $\mathcal{A}$ be an access schema over $\mathcal{R}$. We need to show that if Q is covered by $\mathcal{A}$, then it is also boundedly evaluable under $\mathcal{A}$. We turn the query $Q(\bar{x})$ into an $\mathcal{A}$-equivalent boundedly evaluable query plan in a step-wise way: (1) we first show that there exist query plans that compute a bounded number, yet sufficiently many, values for each of the variables in Q; we do this by simulating the computation of $\mathsf{cov}(Q, \mathcal{A})$; (2) we then leverage the fact that each relation atom in Q is indexed by $\mathcal{A}$ and show that each relation atom in Q can be eliminated and replaced by a relation atom that takes values from a query plan for that relation; (3) finally, we use the standard translation from CQ to SPC to obtain a query plan for Q, guaranteeing $\mathcal{A}$-equivalence along the way.

(1) Individual covered variables. For each variable $u \in \mathsf{cov}(Q,\mathcal{A})$, let $\mathsf{Val}_u(V)$ be a unary relation schema. As a first step, we show that for each covered variable u in $\mathsf{cov}(Q,\mathcal{A})$ we can compute an instance D_u of Val_u, using a boundedly evaluable query plan ξ_u under $\mathcal{A}$, such that for any $D \models \mathcal{A}$, $Q(D) = Q_u(D, D_u)$ where $Q_u(\bar{x}) = \exists \bar{y} P(\bar{x},\bar{y}) \wedge \mathsf{Val}_u(u)$ and $D_u = \xi_u(D)$. In other words, D_u contains a bounded amount of values for the u-variable (its size is determined by the access schema) that are sufficient to answer Q on D. We extend the schema $\mathcal{R}$ with all Val_v relations, one for each covered variable v in Q. Let $\mathcal{R}_1$ be the resulting schema.

To show that ξ_u exists, it suffices to consider a (partial) run of the algorithm for $\mathsf{cov}(Q,\mathcal{A})$ that covers u in the last step. Such a run starts by applying $R(X \to Y, N)$ to an occurrence of R in Q that has constants $\bar{a}$ in its X-attributes. Clearly, $T_1 = \bar{a}$, $T_2 = \mathsf{fetch}(X = \bar{a}, R, Y)$ returns D_y for each y corresponding to an attribute in Y. We simulate further steps of the algorithm in a similar way. For example, if $R(X' \to Y', N')$ is considered, we use $T_1 = \prod_{x \in X'} \xi_{x'}$ to obtain sufficiently many values for the X'-attributes. This is possible since the variables x corresponding to X' must be covered already and each $\xi_{x'}(D)$ returns sufficiently many (bounded in number) values for x'. We then consider $T_2 = \mathsf{fetch}(X' \in T_1, R, Y')$. Observe that $T_2(D)$ indeed provides all necessary values D_y for each y variable corresponding to an attribute Y' in R. We continue in this way until u becomes covered by $\mathcal{A}$. At this point, $D_u = \xi_u(D)$ is obtained.

(2) Relation atoms. Let $U = \{u_1, \ldots, u_k\}$ be a set of variables in Q that are covered by $\mathcal{A}$. Consider the following CQ query over $\mathcal{R}_1$: $Q_U(\bar{x}) = \exists \bar{y} P(\bar{x},\bar{y}) \wedge \bigwedge_{u_i \in U} \mathsf{Val}_{u_i}(u_i)$. It is readily verified that for any $D \models \mathcal{A}$ we have that $Q(D) = Q_U(D, D_{u_1}, \ldots, D_{u_k})$, where $D_{u_i} = \xi_{u_i}(D)$ for the boundedly evaluable query plan ξ_{u_i} ($i \in [1,k]$), computed in step (1).

Consider a relation atom $R(\bar{x},\hat{y},\bar{\bar{z}})$ in Q, distinct from the V_{u_j}'s. Here, variables in $\bar{x}$ are free and covered by $\mathcal{A}$, those in $\hat{y}$ are bound and covered by $\mathcal{A}$, and variables in $\bar{\bar{z}}$ are bound, non-constant and uncovered by $\mathcal{A}$. Since the $\bar{\bar{z}}$ variables can only occur once in Q, because Q is covered, we can push $\exists \bar{\bar{z}}$ to the relation atom $R(\bar{x},\hat{y},\bar{\bar{z}})$. That is, to evaluate Q we do not need values for $\bar{\bar{z}}$ in $R(\bar{x},\hat{y},\bar{\bar{z}})$. Let XY be the attributes corresponding to the covered variables in $\bar{x}$ and $\hat{y}$ in $R(\bar{x},\hat{y},\bar{\bar{z}})$. Surely, for every $D \models \mathcal{A}$, $\pi_{XY}(D) \subseteq \prod_{x_i \in \bar{x}} D_{x_i} \times \prod_{y_i \in \bar{y}} D_{y_i}$.

Note that $R(\bar{x},\hat{y},\bar{\bar{z}})$ is also indexed by $\mathcal{A}$. That is, there exists an access constraint $R(X' \to Y', N) \in \mathcal{A}$ such that X' is included $X \cup Y$. Furthermore, $X' \cup Y'$ covers at least all attributes corresponding to free, constant, or bound variables that occur more than once in Q. In other words, only the attributes corresponding to variables in $\bar{y}$ that occur once in Q may not be indexed by $\mathcal{A}$. We have just seen, however, that the values of such single occurrence variables are not needed to evaluate the query. It is thus safe to consider $\pi_{X'Y'}(D)$ rather than $\pi_{XY}(D)$ when evaluating Q. Let $\mathsf{Val}_R(X',Y')$ be a new relation schema corresponding to the relation atom under consideration, *i.e.,* $R(\bar{x},\hat{\bar{y}},\bar{\bar{z}})$. We extend $\mathcal{R}_1$ with Val_R, resulting in $\mathcal{R}_2$. Let D_R be the instance of Val_R obtained by the following query plan:

$$T_1 = \prod_{x_i \in X'} \xi_{x_i}, T_2 := \mathsf{fetch}(X' \in T, R, Y').$$

That is, $D_R = T_2(D)$ and we denote this query plan also by ξ_R. Then, it is readily verified that for any $D \models \mathcal{A}$, $Q(D) = Q_{U,R}(D, D_{u_1}, \ldots, D_{u_k}; D_R)$ where the latter is a CQ query over $\mathcal{R}_2$ obtained by (i) replacing the relation atom occurrence $R(\bar{x},\hat{\bar{y}},\bar{\bar{z}})$ in Q_U by $\mathsf{Val}_R(\bar{x},\bar{y}')$ where $\bar{x} \cup \bar{y}'$ correspond to the attributes indexed by $R(X' \to Y', N)$; and (ii) removing $R(\bar{x},\hat{\bar{y}},\bar{\bar{z}})$ and the existential quantifiers corresponding to single occurrence variables in $R(\bar{x},\hat{\bar{y}},\bar{\bar{z}})$ that are not covered or indexed.

Observe that we can continue in this way until all relation atoms in Q have been processed. Let $\mathcal{R}'$ be the relational schema at the end of this process. Observe that $\mathcal{R}'$ solely consists $\mathcal{R}_1$ and Val_{R_j}, one for each occurrence of R_j in Q, and this for every $R_j \in \mathcal{R}$.

(3) Complete query. Along the same lines, we have that for any $D \models \mathcal{A}$, $Q(D) = Q_{U,\mathcal{R}'}(D_{u_1}, \ldots, D_{u_k}; D_{R_1}, \ldots, D_{R_\ell})$. Clearly, we can further remove the Val_{u_i}'s from $Q_{U,\mathcal{R}'}$ as we now have a subset of values for u_i as given by the appropriate D_{R_j}. We end up with a CQ Q' over Val_{R_j} relations. Denote by ξ_{R_j} the plans computed in step (2) such that for any $D \models \mathcal{A}$, $D_{R_j} = \xi_{R_j}(D)$. We finally translate Q' into an equivalent SPC query. Clearly, all SPC operations have a direct counter part in query plan operations. Hence, we obtain a query plan ξ by using the ξ_{R_j}'s and in which the relational operators are specified by the equivalent SPC query. It is readily verified that $Q(D) = \xi(D)$ for any $D \models \mathcal{A}$.

Hence, the covered CQ Q is boundedly evaluable under $\mathcal{A}$ by ξ just constructed.

This completes the proof of Lemma 10. □

2.2.3 Bounded Evaluability of UCQ and $\exists\mathsf{FO}^+$

We next study BEP for UCQ and $\exists\mathsf{FO}^+$. While $\mathsf{BEP}(\mathsf{CQ})$ is nontrivial, the presence of union makes the bounded evaluability analysis more intriguing. Recall that for two

UCQ $Q = \bigcup_{i\in[1,m]} Q_i$ and $Q' = \bigcup_{j\in[1,n]} Q'_j$, $Q \sqsubseteq Q'$ iff for each Q_i, there exists Q'_j such that $Q_i \sqsubseteq Q'_j$ [SY80]. This no longer holds when we consider $\sqsubseteq_{\mathcal{A}}$ under an access schema $\mathcal{A}$.

Example 4: Consider a relation schema $R(X)$, an access schema $\mathcal{A}$ with $R(\emptyset \to X, 2)$, and queries below:

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

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

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

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

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

The lemmas below characterize the bounded evaluability of UCQ. They also tell us how to determine whether a query Q in $\exists$FO$^+$is boundedly evaluable, since a query in $\exists$FO$^+$is equivalent to a query in UCQ.

Lemma 11: *Under an access schema $\mathcal{A}$, a* UCQ *Q is boundedly evaluable if and only if Q is $\mathcal{A}$-equivalent to a* UCQ *$Q' = Q_1 \cup \cdots \cup Q_k$ such that for each $i \in [1,k]$, Q_i is boundedly evaluable under $\mathcal{A}$.* □

Proof: Consider a UCQ Q and an access schema $\mathcal{A}$, both defined over a relational schema $\mathcal{R}$. First assume that Q is $\mathcal{A}$-equivalent to a UCQ $Q' = Q_1 \cup \cdots \cup Q_k$ such that each Q_i is boundedly evaluable. Then a boundedly evaluable query plan of Q can be readily generated as the sequence $\xi_1, T_1, \ldots, \xi_k, T_k, S = T_1 \cup T_2, S = S \cup T_3, \ldots, S = S \cup T_k$, by putting together the boundedly evaluable query plans of all those CQ sub-queries of Q_i, followed by union operations.

Conversely, suppose that Q is boundedly evaluable. Then Q has a boundedly evaluable query plan ξ under $\mathcal{A}$ such that it ends up with a sequence of union operations denoting $T_1 \cup \ldots \cup T_k$ (see Section 2.1), where each T_i is computed by a sub-sequence of ξ with $\pi, \sigma_c, \times, \rho$, fetch and constant sets only. From the proof of Lemma 8 it follows that T_i is a bounded query plan for a CQ Q_i, *i.e.,* Q_i is boundedly evaluable under $\mathcal{A}$. Moreover, $Q \equiv_A Q_1 \cup \ldots Q_k$. □

Let the element expansion of a UCQ (reps. $\exists$FO$^+$) be the union of element expansions of its CQ sub-queries. With Lemma 11, we have the following directly characterization on boundedly evaluable UCQ and $\exists$FO$^+$queries.

Lemma 12: *A* UCQ *Q is boundedly evaluable iff in its element expansion $Q^e = Q^e_1 \cup \ldots \cup Q^e_m$, for each Q^e_i, (a) either Q^e_i is boundedly evaluable, or (b) for all valuations θ of the tableau (T_i, u_i) of Q^e_i, there exists an element sub-query Q^e_j in Q^e such that $\theta(u_i) \in Q^e_j(\theta(T_i))$ and Q^e_j is boundedly evaluable.* □

Proof: First assume that Q is boundedly evaluable. Then Q is $\mathcal{A}$-equivalent to a UCQ Q' in which each CQ sub-query is boundedly evaluable, by Lemma 11. Let $Q^{e'} = Q^{e'}_1 \cup \ldots \cup Q^{e'}_n$ be the element expansion of Q'. Then one can verify that each $Q^{e'}_j$ is boundedly evaluable and $T'_j \models \mathcal{A}$, following Lemma 11. Since $Q^{e'} \equiv_{\mathcal{A}} Q \equiv_{\mathcal{A}} Q^e$, we have that $Q^{e'} \equiv Q^e$ with the standard definition of query equivalence since the tableau of each element query satisfies $\mathcal{A}$, by following Lemma 7. Consider minimized Q^e, denoted by Q^e_M, *i.e.,* no sub-query in Q^e is contained in another; similarly for $Q^{e'}_M$, Then $Q^{e'}_M \equiv Q^e_M$, and each sub-query of Q^e_M is equivalent to a sub-query of $Q^{e'}_M$ [SY80], and is hence boundedly evaluable. Hence for each element sub-query Q^e_i of Q^e (before minimization), either Q^e_i is already boundedly evaluable under $\mathcal{A}$, or for any valuation θ of the tableau (T_i, u_i) of Q^e_i, there exists Q^e_j in Q^e such that $\theta(u_i) \in Q^e_j(\theta(T_i))$ and Q^e_j is boundedly evaluable, *i.e.,* Q^e_i is "absorbed" in the minimization process.

Conversely, if the condition holds, let Q^e_b be the union of all boundedly evaluable element sub-queries of Q^e. Then $Q^e_b \equiv Q^e \equiv_{\mathcal{A}} Q$. By Lemma 11, Q is boundedly evaluable under $\mathcal{A}$. □

With the characterization, below we show that BEP is decidable for UCQ and $\exists$FO$^+$.

Corollary 13: BEP *is* EXPSPACE-*hard and in* 2EXPSPACE *for* $\exists$FO$^+$. □

Proof: The lower bound follows from Theorem 3. For the upper bound, we give a 2NEXPSPACE (2EXPSPACE) algorithm for checking BEP($\exists$FO$^+$), based on Lemma 12. We give an EXPSPACE algorithm for checking whether a UCQ Q is *not* boundedly evaluable under $\mathcal{A}$, as follows.

(a) Guess an element sub-query Q_i^m.

(b) Check whether Q_i^m is boundedly evaluable. If so, reject the guess and repeat the process. Otherwise continue.

(c) Check whether there exists a valuation θ of (T_i, u_i) of Q_i^e that is not contained in all boundedly evaluable sub-queries Q_j^e of Q^e. This can be done in 2EXPSPACE (by combining the guess of θ with the guess of Q_i^m). If so, return "yes", and continue otherwise.

Note that steps (b) and (c) are in 2EXPSPACE. Hence the algorithm is in 2NEXPSPACE = 2EXPSPACE. Its correctness is ensured by Lemma 12. Therefore, BEP is in 2EXPSPACE for UCQ.

Observe that the same algorithm also works when Q is in $\exists$FO$^+$, since Q is equivalent to a UCQ. Hence BEP is also in 2EXPSPACE for $\exists$FO$^+$. □

Summary

We have investigated how to query big data by leveraging bounded evaluability, to compute exact answers by accessing a bounded amount of data. We have identified fundamental problems associated with bounded evaluability, and provided their complexity and characterizations.

Chapter 3

On the Effective Syntax of Bounded Evaluability

In the last chapter, we have formalized and characterized bounded evaluability. It is undecidable to determine whether a query in relational algebra (RA) is bounded under an access schema $\mathcal{A}$. In light of the undecidability, this chapter develops an effective syntax for bounded RA queries. We identify a class of *covered* RA queries such that under $\mathcal{A}$, (a) every boundedly evaluable RA query is equivalent to a covered query, (b) every covered RA query is boundedly evaluable, and (c) it takes PTIME in $|Q|$ and $|\mathcal{A}|$ to check whether Q is covered by $\mathcal{A}$. We provide quadratic-time algorithms to check the coverage of Q, and to generate a bounded query plan for covered Q. We also study a new optimization problem for minimizing access constraints for covered queries and develop its complexity and algorithms. These provide us with both fundamental results and practical algorithms for making use of bounded evaluability. Using real-life data, we experimentally verify that a large number of RA queries in practice are covered, and that bounded query plans improve RA query evaluation by orders of magnitude.

We start with an example that shows the connections between RA query equivalence and bounded evaluability.

Example 5: Consider an example query Q_0 from Graph Search of Facebook [Faca]: *find me all restaurants in* NYC *which I have not been to, but in which my friends have dined in May, 2015*. The query is posed on dataset D_0, which consists of three relations: (a) $\mathsf{friend}(\mathsf{pid}, \mathsf{fid})$, stating that fid is a friend of pid, (b) $\mathsf{dine}(\mathsf{pid}, \mathsf{cid}, \mathsf{month}, \mathsf{year})$ indicating that a person pid dined in restaurant cid in month of year, and (c) $\mathsf{cafe}(\mathsf{cid}, \mathsf{city})$, stating that cid is located in city.

Query Q_0 is given in RA, with constant p_0 denoting “me”:

$$\begin{aligned}
Q_0(\mathsf{cid}) &= Q_1(\mathsf{cid}) - Q_2(\mathsf{cid}), \text{ where}\\
Q_1(\mathsf{cid}) &= \pi_{\mathsf{cid}}\big(\mathsf{friend}(p_0, \mathsf{fid}) \bowtie_{\mathsf{fid}=\mathsf{pid}} \mathsf{dine}(\mathsf{pid}, \mathsf{cid}, \text{MAY}, 2015)\\
&\qquad \bowtie_{\mathsf{cid}=\mathsf{cid}'} \mathsf{cafe}(\mathsf{cid}', \text{NYC})\big), \text{ and}\\
Q_2(\mathsf{cid}) &= \pi_{\mathsf{cid}}\ \mathsf{dine}(p_0, \mathsf{cid}, \mathsf{month}, \mathsf{year}).
\end{aligned}$$

Dataset D_0 may be big, with billions of users and trillions of friend links [GBDS14]. It is costly to compute $Q_0(D_0)$ directly.

Nonetheless, a closer examination of D_0 reveal a set $\mathcal{A}_0$ of access constraints:

- ψ_1: $\mathsf{friend}(\mathsf{pid} \to \mathsf{fid}, 5000)$;
- ψ_2: $\mathsf{dine}((\mathsf{pid}, \mathsf{year}, \mathsf{month}) \to \mathsf{cid}, 31)$;
- ψ_3: $\mathsf{dine}((\mathsf{pid}, \mathsf{cid}) \to (\mathsf{pid}, \mathsf{cid}), 1)$;
- ψ_4; $\mathsf{cafe}(\mathsf{cid} \to \mathsf{city}, 1)$.

Here ψ_1 specifies a constraint imposed by Facebook [Facb]: a limit of 5000 friends per user; ψ_2 states that each person dines in at most 31 restaurants each month; ψ_3 says that (pid, cid) is a “key” of the pair, and ψ_4 states that each restaurant id is associated with a single city. Indices can be built on D_0 based on ψ_1 such that given a person, it returns all the ids of her friends by accessing at most 5000 friend tuples; similarly for ψ_2, ψ_3 and ψ_4.

Given the access constraints, we can compute $Q_1(D_0)$ by accessing at most 315000 tuples from D_0, instead of trillions. (1) We identify and fetch T_1 of at most 5000 fid’s of friend tuples with $\mathsf{pid} = p_0$, by using the index built for ψ_1. (2) For each fid value f in T_1, we fetch T_2 of at most 31 cid’s of dine tuples with $\mathsf{fid} = f$, $\mathsf{year} = 2015$ and $\mathsf{month} = \text{MAY}$, leveraging the index for ψ_2. (3) For each cid in T_2, we fetch its cafe tuple by using the index for ψ_4, and return a set T_3 of cid’s from these tuples with $\mathsf{city} = \text{NYC}$. The query plan fetches at most 5000 + 5000 $\times$ 31 $\times$ 2 tuples only, all using indices, to compute $Q_1(D_0)$ no matter how big D_0 is. Therefore, Q_1 is boundedly evaluable under $\mathcal{A}_0$.

However, query Q_2 is not bounded under $\mathcal{A}_0$: we cannot make use of any indices above when accessing the (possibly huge) dine relation given pid = p_0 alone. Since the set difference operator in Q_0 forces us to check *all* tuples in $Q_2(D_0)$, one might think that Q_0 is not bounded either.

Nonetheless, observe that Q_0 is equivalent to $Q_0'(\mathsf{cid}) = Q_1(\mathsf{cid}) - Q_3(\mathsf{cid})$, where $Q_3(\mathsf{cid}) = Q_1(\mathsf{cid}) \bowtie_{\mathsf{cid}=\mathsf{cid}'} Q_2(\mathsf{cid}')$. Moreover, Q_3 is boundedly evaluable. Indeed, for each cid value returned by $Q_1(D_0)$ (*i.e.,* T_3 above), we can check whether (p_0, cid) is a pair occurring in relation dine, by accessing one tuple via the index for ψ_3. We return all those cid's that pass the check. Thus we can answer $Q_3(D_0)$ by accessing 5000×31 tuples. Therefore, Q_0 is equivalent to bounded Q_0', with a query plan consisting of the plan for Q_1 above, followed by the plan for Q_3; it accesses at most 470000 tuples only, no matter how big D_0 grows. This shows that Q_0 is actually boundedly evaluable under $\mathcal{A}_0$. Indeed, a boundedly evaluable query plan for Q_0 under $\mathcal{A}_0$ is as follows.

$$
\begin{aligned}
&T_1 = \{p_0\},\ T_2 = \mathsf{fetch}(T_1, \mathsf{friend}, \mathsf{fid}),\ T_3 = \pi_{\mathsf{fid}}(T_2),\\
&T_4 = \{2015\},\ T_5 = \{\textsc{may}\},\ T_6 = T_4 \times T_5,\ T_7 = T_3 \times T_6,\\
&T_8 = \mathsf{fetch}(X \in T_7, \mathsf{dine}, \mathsf{cid}),\ T_9 = \pi_{\mathsf{cid}}(T_8),\\
&T_{10} = \mathsf{fetch}(X \in T_9, \mathsf{cafe}, \mathsf{city}),\ T_{11} = \sigma_{\mathsf{city}=\textsc{nyc}}(T_{10}),\\
&T_{12} = \pi_{\mathsf{cid}}(T_{11}),\\
&T_{13} = T_1 \times T_{12},\ T_{14} = \mathsf{fetch}(X \in T_{13}, \mathsf{dine}, (\mathsf{pid}, \mathsf{cid})),\\
&T_{15} = \pi_{\mathsf{cid}}(T_{14}),\ T_{16} = T_{13} \setminus T_{15}.
\end{aligned}
$$

Note that the sequence $T_1, \ldots, T_{12}$ forms a boundedly evaluable query plan for sub-query Q_1 of Q_0 under $\mathcal{A}_0$.

Consider another access schema $\mathcal{A}_1$ and RA query Q_4 defined on relation schemas $R(A,B,E)$ and $S(F,G,H)$:

- ∘ $\mathcal{A}_1 = \{R(AB \to E, N), S(F \to GH, 2), S(GH \to GH, 1)\}$.
- ∘ $Q_4 = Q_4^1 - Q_4^2$, where $Q_4^1 = \pi_x(R(1,x,y) \bowtie S(w,x,y) \bowtie S(w,1,x) \bowtie S(w,x,x))$ and $Q_4^2 = \pi_x(R(1,x,x) \bowtie S(u,1,x) \bowtie S(u,x,x))$, where $\bowtie$ denotes natural join.

At a first glance, Q_4 seems not boundedly evaluable, since we cannot retrieve x and w values using indices in $\mathcal{A}_1$ and thus cannot get y for Q_4^1. Similarly, we cannot get u and x values for Q_4^2. However, under $S(F \to GH, 2)$ in $\mathcal{A}_1$, observe that (x,y) must be equal to either $(1,x)$ or (x,x) in all tuples retrieved from instance of S by any query plan for Q_4^1. In other words, under $\mathcal{A}_1$, the SPC sub-query Q_4^1 reduces to SPCU $Q_4^{1'} \cup Q_4^{1''}$, where $Q_4^{1'} = \pi_x(R(1,1,x) \bowtie S(w,1,x) \bowtie S(w,x,x)$ and $Q_4^{1''} = Q_4^2$. Thus, under $\mathcal{A}_1$, Q_4 is equivalent to $Q_4^{1'}$, which is boundedly evaluable under $\mathcal{A}_1$. □

The example tells us that to decide whether an RA (SQL) query is bounded, it is often necessary to check query equivalence (*e.g.,* Q_0 and Q_0'), which is undecidable for RA queries in the presence of set difference [AHV95]. Furthermore, the analysis is further complicated by access constraints when union ($\cup$) and set difference ($-$) are present. Indeed, the presence of union allows us to convert SPC to SPCU under $\mathcal{A}$ (*e.g.,* Q_4^1 to $Q_4^{1'} \cup Q_4^{1''}$), which may further interact with set difference and can be nontrivially made bounded (*e.g.,* Q_4 and $Q_4^{1'}$).

A natural question is **Q2** in Chapter 1, which asks whether it is still possible to make practical use of bounded evaluability of RA queries, given the intractability?

Overview. This chapter is to answer the question. We approach it by identifying an *effective syntax* for boundedly evaluable RA queries. That is, a class $\mathcal{L}$ of RA queries such that under a set $\mathcal{A}$ of access constraints,

(a) every boundedly evaluable RA query is equivalent to a query in $\mathcal{L}$, *i.e.,* $\mathcal{L}$ expresses all bounded RA queries;

(b) every query Q in $\mathcal{L}$ is boundedly evaluable; and

(c) it takes PTIME (polynomial time) in $|Q|$ and $|\mathcal{A}|$ to syntactically check whether Q is in $\mathcal{L}$.

That is, $\mathcal{L}$ identifies the *core* subclass of boundedly evaluable RA queries, *without sacrificing their expressive power*.

The study of bounded evaluability is analogous, to an extent, to the study of safe relational calculus queries, which are also undecidable. Effective syntax was first studied 30 years ago [Ull82, ST95, GT91], to express all safe queries up to equivalence. It is now underlying SQL and commercial DBMS (*e.g.,* Microsoft Access [AHV95]), to ensure that input queries Q are safe by, *e.g.,* enforcing a range for each variable in Q. It imposes syntactical restrictions on undecidable safe queries, such that the restricted class is efficiently decidable.

Along the same lines, effective syntax allows us to make practical use of bounded evaluability. (1) It provides us with a guideline for formulating bounded evaluable queries, just like its counterpart for safe queries. (2) As will be shown shortly, bounded evaluability analysis can be readily incorporated into commercial DBMS. Given an input RA query Q, it first checks whether Q is in $\mathcal{L}$, in PTIME by condition (c) above; if so, it generates a bounded query plan for Q by using indices in $\mathcal{A}$, which is warranted to exist by (b). (3) By (a), if Q is boundedly evaluable, it can be expressed in $\mathcal{L}$. Hence query rewriting rules can be implemented to transform Q to an equivalent query in $\mathcal{L}$, to an extent.

More specifically, we provide theoretical results and practical methods for the bounded evaluability of RA as follows.

(1) We develop an effective syntax $\mathcal{L}$ for boundedly evaluable RA queries (Section 3.1), referred to as *covered* queries. In a nutshell, an RA query Q is covered if for any relation in Q, its attributes *needed for answering* Q can be fetched via the indices in $\mathcal{A}$, in time bounded by the cardinality constraints of $\mathcal{A}$. We prove that every boundedly evaluable RA query under $\mathcal{A}$ is also covered by $\mathcal{A}$ (*i.e.,* property (a)).

(2) We develop an algorithm for checking covered queries (Section 3.2). Given an RA query Q and a set $\mathcal{A}$ of access constraints, the algorithm decides whether Q is covered by $\mathcal{A}$ in $O(|Q|^2 + |\mathcal{A}|)$-time, where $|Q|$ is the size of Q and $|\mathcal{A}|$ is the total length of access constraints in $\mathcal{A}$, independent of the size $|D|$ of dataset D. In practice, $|Q|$ and $|\mathcal{A}|$ are typically much smaller than $|D|$. This proves property (c).

(3) We provide an algorithm to generate query plans for covered queries (Section 3.3). Given an RA query Q covered by $\mathcal{A}$, the algorithm generates a query plan ξ of length $O(|Q||\mathcal{A}|)$ such that for any dataset D that satisfies $\mathcal{A}$, ξ computes $Q(D)$ by accessing a bounded amount of data determined by Q and $\mathcal{A}$. The algorithm is based on a non-trivial characterization of covered RA queries and takes $O(|Q|(|Q| + |\mathcal{A}|))$ time, again independent of $|D|$. This proves property (b).

(4) We also study a new optimization problem (Section 3.4). Given a query Q covered by $\mathcal{A}$, it is to find a subset $\mathcal{A}_m \subseteq \mathcal{A}$ such that Q remains covered by $\mathcal{A}_m$ and the estimated data access via $\mathcal{A}_m$ is minimized. We show that the problem is NP-complete and is not in APX, *i.e.,* it has no PTIME constant-factor approximation algorithm. Nonetheless, we develop efficient heuristic algorithms with performance guarantees, some with reasonable approximation bounds.

(5) We show how bounded evaluability analysis can be integrated into existing DBMS (Section 3.5). Given an RA query Q and a set $\mathcal{A}$ of access constraints, we check whether Q is covered by $\mathcal{A}$, and if so, we generate a bounded query plan for Q with minimal constraints in $\mathcal{A}$, and compute $Q(D)$ by accessing a small fraction D_Q of D, all by using the algorithms described above. We also show how access constraints can be discovered and incrementally maintained.

(6) We implement our approach on top of MySQL and PostgreSQL and experimentally

evaluate its effectiveness using two real-life datasets and a commercial benchmark (query templates and datasets; Section 3.6). We find the following on the real-life data: under a set $\mathcal{A}$ of at most 266 access constraints, on average (a) 67.5% of randomly generated RA queries are boundedly evaluable, among which 83.5% are covered; (b) our query plans outperform MySQL and PostgreSQL that use the same indices by at least *3 orders of magnitude* (see more in Section 3.6 for details about the configurations and optimziations enabled for MySQL and PostgreSQL), and the gap gets larger on bigger data; (c) our plans access only 0.0019% of the data; that is, they "reduce" D from PB to GB; and (d) the indices account for 14.8% of the original data. We also find that (e) our algorithms for coverage checking, plan generation and minimizing access constraints are all efficient: they take at most 199ms in all cases.

These results settle the open question for the study of RA boundedly evaluability, from theory to practice. They suggest an approach to answering queries within bounded resources, by adding the functionality of bounded evaluation to existing DBMS. It is a common practice for decades in query evaluation to access as little data as possible, rather than the entire dataset, by making use of various indices. This work is an effort to formalize the idea, to decide when it is feasible to answer a query within bounded resources, and to provide a systematic method to achieve it.

3.1 Effective Syntax for Bounded Evaluability

Below we present an effective syntax $\mathcal{L}$ for boundedly evaluable RA queries, referred to as the class of *covered queries*.

To simplify the exposition, we consider RA queries Q in a *normal form* in which all occurrences of each relation name are made distinct via renaming. For an access constraint $\phi = R(X \to Y, N)$ and a renaming S of R in Q, we refer to $S(X \to Y, N)$ as the *actualized constraint* of ϕ on S, and to the set of all actualized constraints of $\mathcal{A}$ as the *actualized access schema* of $\mathcal{A}$ on Q. We consider *w.l.o.g.* normalized Q and actualized $\mathcal{A}$ only, based on the lemma below.

Lemma 14: *Given any* RA *query Q and access schema $\mathcal{A}$ over relational schema $\mathcal{R}$, one can compute the actualized access schema $\mathcal{A}'$ from Q and $\mathcal{A}$ in $O(|Q||\mathcal{A}|)$-time such that*

(1) for any instance D of $\mathcal{R}$, $D \models \mathcal{A}$ iff $D \models \mathcal{A}'$; and

(2) Q is boundedly evaluable under $\mathcal{A}$ iff Q' is boundedly evaluable under $\mathcal{A}'$ (iff for

if and only if). □

Covered queries. We now define covered queries, starting with SPC. Intuitively, an SPC query Q is covered if for any relation S in Q, all the attributes of S needed to answer Q can be fetched via indices in $\mathcal{A}$ and moreover, their sizes are bounded by the cardinality constraints of $\mathcal{A}$.

Consider an SPC query $Q = \pi_Z \sigma_C(S_1 \times \ldots \times S_n)$ defined over a relational schema $\mathcal{R}$, where Z is a set of attributes of $\mathcal{R}$, C is the selection condition of Q, and S_i's are distinct relations after renaming (Lemma 14). We use Σ_Q to denote the set of all equality atoms $A = A'$ or $A = c$ derived from C by the transitivity of equality. For any sets X and X' of attributes of Q, we write $\Sigma_Q \vdash X = X'$ if $X = X'$ can be derived from Σ_Q, which can be checked in $O(\mathsf{max}(|X|, |X'|))$ time (after an $O(|Q|^2)$-time preprocessing of Q).

Coverage. The *set of covered attributes* of Q by an access schema $\mathcal{A}$, denoted by $\mathsf{cov}(Q, \mathcal{A})$, includes attributes that can be accessed via indices in $\mathcal{A}$. It is defined as follows:

- if $\Sigma_Q \vdash \sigma_{A=c}$, then $A \in \mathsf{cov}(Q, \mathcal{A})$;
- if $R(\emptyset \to X, N) \in \mathcal{A}$, then $R[X] \subseteq \mathsf{cov}(Q, \mathcal{A})$;
- if $R[X] \subseteq \mathsf{cov}(Q, \mathcal{A})$ and $\Sigma_Q \vdash R[X] = S[Y]$, then $S[Y] \subseteq \mathsf{cov}(Q, \mathcal{A})$; and
- if $R(X \to Y, N) \in \mathcal{A}$ and $R[X] \subseteq \mathsf{cov}(Q, \mathcal{A})$, then $R[Y] \subseteq \mathsf{cov}(Q, \mathcal{A})$.

Here $R(\emptyset \to X, N)$ is an access constraint stating that there are at most N distinct X values in an instance of R, *e.g.,* there exist at most 12 distinct months per year.

Covered SPC. Denote by X_Q the set of attributes in an SPC query Q that occur in either its selection condition C or the projection attributes Z of Q. We say that Q is

- *fetchable via* $\mathcal{A}$ if $X_Q \subseteq \mathsf{cov}(Q, \mathcal{A})$; and
- *indexed by* $\mathcal{A}$ if for each relation name S in Q, there is an actualized constraint $S(X \to Y, N)$ of $\mathcal{A}$ such that
  - $S[X] \subseteq \mathsf{cov}(Q, \mathcal{A})$, and
  - $S[XY]$ includes all attributes of S that are in X_Q, *i.e.,* attributes XY come from the same tuple.

An SPC query Q is *covered by* $\mathcal{A}$ if Q is both fetchable via $\mathcal{A}$ and indexed by $\mathcal{A}$. That is, all attributes needed by Q can be fetched using indices of $\mathcal{A}$ and are bounded by $\mathcal{A}$.

Covered RA. We represent an RA query Q as its *query (syntax) tree* T^Q [AHV95]. To simplify the discussion, we say that an RA query Q' is a *sub-query* of Q if $T^{Q'}$ is a sub-tree of T^{Q}.

A *max* SPC *sub-query* of Q is a sub-query Q_s such that

- Q_s is an SPC query, and
- there exists no sub-query Q'_s of Q such that it is also in SPC, $Q_s \neq Q'_s$, and Q_s is a sub-query of Q'_s.

An RA query Q is *covered by an access schema $\mathcal{A}$* if for all max SPC sub-queries Q_s of Q, Q_s is covered by $\mathcal{A}$. Similarly, Q is *fetchable via $\mathcal{A}$* (resp. *indexed by $\mathcal{A}$*) if each max sub-SPC sub-query is fetchable via $\mathcal{A}$ (resp. indexed by $\mathcal{A}$).

Intuitively, an RA query Q is "normalized" by pushing set difference to the top level, on (unions of) max SPC sub-queries. These max SPC sub-queries characterize all relation attributes that need to be accessed when answering Q.

Example 6: For the queries and $\mathcal{A}_0$ of Example 5, Q_1 and Q_3 are covered by $\mathcal{A}_0$, but Q_2 is not. Indeed, $X_{Q_1} = \{x_{p_0}, \mathsf{fid}, \mathsf{pid}, \mathsf{cid}, x_{\mathrm{MAY}}, x_{2015}, \mathsf{cid}', x_{\mathrm{NYC}}\} = \mathsf{cov}(Q_1, \mathcal{A}_0)$, where x_d denotes the attribute corresponding to a constant d in Q_1. Hence Q_1 is fetchable via $\mathcal{A}_0$; moreover, Q_1 is indexed by $\mathcal{A}_0$ since friend, dine and cafe are indexed by ψ_1, ψ_2 and ψ_4, respectively; similarly for Q_3. However, Q_2 is not fetchable via $\mathcal{A}_0$ since $\mathsf{cov}(Q_2, \mathcal{A}_0) = \{x_{p_0}\}$ but $X_{Q_2} = \{x_{p_0}, \mathsf{cid}\}$, and relation dine is not indexed by any constraint in $\mathcal{A}_0$ for Q_2. As a result, Q'_0 is covered by $\mathcal{A}_0$ since both of its max SPC sub-queries Q_1 and Q_3 are covered by $\mathcal{A}_0$. In contrast, Q_0 is not covered by $\mathcal{A}_0$ since Q_2 is not. □

The main result of the chapter is as follows.

Theorem 15: *Under access schema $\mathcal{A}$, for any* RA *query Q,*

(1) if Q is boundedly evaluable under $\mathcal{A}$, then Q is $\mathcal{A}$-equivalent to an RA *query Q' that is covered by $\mathcal{A}$;*

(2) if Q is covered, then Q is boundedly evaluable; and

(3) it takes PTIME *to check whether Q is covered by $\mathcal{A}$.* □

That is, we reduce the problem of deciding RA bounded evaluability to syntactic checking of covered queries, *without losing the expressive power*. Indeed, all boundedly evaluable RA queries have an $\mathcal{A}$-equivalent covered version. For these RA queries, covered queries play the same role as range-safe RA queries for checking "the safety" SQL queries [AHV95].

Proof: Theorem 15(3) is verified by the syntactic definition of covered RA queries.

Below we focus on (1) and (2).

Proof of Theorem 15(1). It suffices to show that for any boundedly evaluable query plan ξ under $\mathcal{A}$, there exists a covered RA query Q_ξ such that $Q_\xi \equiv_{\mathcal{A}} \xi$. We show this by proving a stronger result:

(1) for any boundedly evaluable query plan ξ under $\mathcal{A}$, there exists an RA query Q'_ξ such that Q'_ξ is $\mathcal{A}$-equivalent to ξ and Q'_ξ is *strongly covered* by $\mathcal{A}$; and

(2) an RA query Q is covered by $\mathcal{A}$ if it is strongly covered by $\mathcal{A}$.

We next define strongly covered queries and show the result.

Strong normal form. An RA query Q is in a *strong normal form* if it is in the normal form and moreover, in its syntax tree T^Q, for each set operator γ, *i.e.,* union $\cup$ or set-difference $-$, there exist no selection or Cartesian-product operators on the path from γ to the root of T_Q. Intuitively, the selections and Cartesian-products are pushed through set-difference and union in queries of strong normal form. It should be remarked that this is not a must. Instead, it is to simplify the proof. Moreover, notice that projections cannot be pushed through either set-difference.

It suffices to consider queries in the strong normal form only.

Lemma 16: *For any* RA *query Q over a relational schema $\mathcal{R}$, there exists an* RA *query Q' over $\mathcal{R}$ such that $Q' \equiv Q$ and Q' is in the strong normal form.* □

Proof of Lemma 16. We translate Q to Q' inductively based on the structure of Q. For an sub-expression E of Q, we use E' to denote the corresponding translation in Q':

(a) if E is R for some $R \in \mathcal{R}$, then E' is R;

(b) if E is $\{\bar{c}\}$ for some constant $\bar{c}$, then E' is $\{\bar{c}\}$;

(c) if E is $\sigma_C(E_1)$: (i) if E_1 is in SPC, then E' is $\sigma_C(E'_1)$; (ii) if E_1 is $E_1^1 - E_1^2$, then E' is $\sigma_C((E_1^1)')$ - $\sigma_C((E_1^2)')$; (iii) if E_1 is $E_1^1 \cup E_1^2$, then E' is $\sigma_C((E_1^1)') \cup \sigma_C((E_1^2)')$;

(d) if E is $\pi_F(E_1)$, then $E' = \pi_F(E'_1)$.

(e) when E is $E_1 \times E_2$: (i) if both E_1 and E_2 are SPC queries, then E is $E'_1 \times E'_2$; (ii) if E_1 (resp. E_2) is $E_{11} - E_{12}$ (resp. $E_{21} - E_{22}$), then E' is $(E_{11} \times E_2)' - (E_{12} \times E_2)'$ (resp. $(E_1 \times E_{21})' - (E_1 \times E_{22})'$), based on laws for set algebras [Sto61]; (iii) similarly for $E_1 \cup E_2$; and

(f) if E is $E_1 - E_2$ (resp. $E_1 \cup E_2$), then E' is $E'_1 - E'_2$ (resp. $E'_1 \cup E'_2$).

One can readily verify that $Q' \equiv Q$. □

Strongly covered queries. An RA query Q is *strongly covered* by an access schema $\mathcal{A}$

if Q is covered by $\mathcal{A}$ and moreover, Q is in the strong normal form.

Obviously a strongly covered query is also a covered query. Therefore, we just need to show statement (1) above. To show (1), we construct an RA query Q_ξ from ξ by replacing every $\mathsf{fetch}(X \in T_j, R, Y)$ operation in ξ with $\pi_{S[XY]}\sigma_{Y_{Q_j}=S[X]}(Q_j \times R(X,Y,Z))$, where Q_j is the rewriting of the first j operations $T_1 = \delta_1, \ldots, T_j = \delta_j$ of ξ, and Y_{Q_j} is the set of attributes of the output relation of Q_j. By the semantics of $\mathsf{fetch}(X \in T_j, R, Y)$, $Q_\xi \equiv_{\mathcal{A}} \xi$.

We next show that there exists an RA query Q'_ξ that is strongly covered by $\mathcal{A}$ such that $Q'_\xi \equiv Q_\xi$ (and thus, $Q'_\xi \equiv_{\mathcal{A}} \xi$), by induction on the length $|\xi|$ of ξ.

<u>*Basis*</u>. When $|\xi| = 1$, Q_ξ can only be one of the following: (a) $\{a\}$ or (b) $\mathsf{fetch}(X \in \emptyset, R, Y)$. In both of the cases, let $Q'_\xi = Q_\xi$. Then Q'_ξ is strongly covered by $\mathcal{A}$.

<u>*Induction step*</u>. As induction hypothesis, assume that for any boundedly evaluable query plan ξ under $\mathcal{A}$ with $|\xi| \leqslant k$ ($k \geqslant 1$), there exists $Q'_\xi \equiv_{\mathcal{A}} Q_\xi$ such that Q'_ξ is strongly covered by $\mathcal{A}$. Consider ξ: $T_1 = \delta_1, \ldots, T_{k+1} = \delta_{k+1}$ with $|\xi| = k+1$. We next show that there exists a strongly covered Q'_ξ such that Q'_ξ is $\mathcal{A}$-equivalent to Q_ξ of ξ, by distinguishing the following cases of the last step $T_{k+1} = \delta_{k+1}$ of ξ.

(1) When δ_{k+1} is $\{a\}$. Let Q'_ξ be $\{a\}$. Then $Q'_\xi \equiv Q_\xi \equiv_{\mathcal{A}} \xi$ and Q'_ξ is strongly covered by $\mathcal{A}$.

(2) When δ_{k+1} is $\pi_Y(T_j)$ for $j < k+1$. By the induction hypothesis, there exists Q'_{ξ_j} that is strongly covered by $\mathcal{A}$ and moreover, $Q'_{\xi_j} \equiv_{\mathcal{A}} Q_{\xi_j}$ of ξ_j, where ξ^j is a prefix of ξ from T_1 to T_j. Then let Q'_ξ be $\pi_Y(Q'_{\xi_j})$. Obviously, $Q'_\xi \equiv Q_\xi \equiv_{\mathcal{A}} \xi$. Moreover, Q'_ξ is strongly covered by $\mathcal{A}$ as Q'_{ξ_j} is.

(3) When δ_{k+1} is $\sigma_C(T_j)$ for $j < k+1$. By the induction hypothesis, there exists Q'_{ξ_j} that is strongly covered by $\mathcal{A}$ and is equivalent to Q_{ξ_j} of ξ_j, which is a prefix $T_1, \ldots, T_j$ of ξ. Note that $\sigma_C(Q'_{\xi_j}) \equiv_{\mathcal{A}} Q_\xi$ and is covered by $\mathcal{A}$, but it may not be strongly covered. We show that there exists Q'_ξ that is equivalent to Q_ξ and is strongly covered by $\mathcal{A}$. Assume *w.l.o.g.* that $Q'_{\xi_j} = Q_1 - Q_2$, where Q_1 and Q_2 are strongly covered queries that are $\mathcal{A}$-equivalent to prefixes ξ^1 and ξ^2 of ξ_j, respectively. The proof for other cases are similar. Let Q' be $\sigma_C(Q_1) - \sigma_C(Q_2)$. Observe the following: (i) Q' is well-defined; and (ii) $Q' \equiv Q_\xi$. As $|\xi^1| \leqslant k$ and $|\xi^2| \leqslant k$, by the induction hypothesis, there exist Q'_1 and Q'_2 such that $Q'_i \equiv_{\mathcal{A}} \sigma_C(Q_i)$ and Q'_i is strongly covered (for $i = 1, 2$). Let Q'_ξ be $Q'_1 - Q'_2$ (or $Q'_1 \cup Q'_2$). Then we have $Q'_\xi \equiv Q' \equiv_{\mathcal{A}} \xi$ and Q'_ξ is strongly covered by $\mathcal{A}$.

(4) When δ_{k+1} is $T_i \times T_j$ for $i, j < k+1$. Similar to case (3) above, by the induction

hypothesis, there exist strongly covered queries Q'_{ξ_i} and Q'_{ξ_j} for prefixes $\xi_i = T_1, \ldots, T_i$ and $\xi_j = T_1, \ldots, T_j$ of ξ, such that $Q'_{\xi_i} \equiv_{\mathcal{A}} \xi_i$ and $Q'_{\xi_j} \equiv_{\mathcal{A}} \xi_j$. So $\xi \equiv_{\mathcal{A}} Q'_{\xi_i} \times Q'_{\xi_j}$. One can prove that there exists $Q'_{\xi} \equiv Q'_{\xi_i} \times Q'_{\xi_j}$ by induction on the structure of Q'_{ξ_i}. As an example, we give details for the case when $Q'_{\xi_i} = Q_1 - Q_2$ and $Q'_{\xi_j} = Q_3 - Q_4$, where Q_1 and Q_2, Q_3 and Q_4 are strongly covered queries for prefixes of ξ_i and ξ_j, respectively. Let Q'_{ξ} be $Q_1 \times Q_3 - Q_2 \times Q_3 - Q_1 \times Q_4$. By rewriting laws of set algebra [Sto61], we know that $Q'_{\xi} \equiv Q'_{\xi_i} \times Q'_{\xi_j}$, and moreover, Q'_{ξ} is strongly covered by the induction hypothesis.

(5) When δ_{k+1} is $T_i \cup T_j$ or $T_i \setminus T_j$ for $i, j < k+1$. The proof is similar to (yet simpler than) case (4) above. By the induction hypothesis, there exist strongly covered queries Q'_{ξ_i} and Q'_{ξ_j} for prefixes $\xi_i = T_1, \ldots, T_i$ and $\xi_j = T_1, \ldots, T_j$ of ξ such that $Q'_{\xi_i} \equiv_{\mathcal{A}} \xi_i$ and $Q'_{\xi_j} \equiv_{\mathcal{A}} \xi_j$. Let $Q'_{\xi} = Q'_{\xi_i} \cup Q'_{\xi_j}$ (or $Q'_{\xi_i} - Q'_{\xi_j}$). Then $\xi \equiv_{\mathcal{A}} Q'_{\xi}$ and Q'_{ξ} is already strongly covered by $\mathcal{A}$ since both Q'_{ξ_i} and Q'_{ξ_j} are.

(6) When δ_{k+1} is $\mathsf{fetch}(X \in T_j, R, Y)$ *for $j < k+1$.* Let ξ_j be the prefix $T_1 = \delta_1, \ldots, T_j = \delta_j$ of ξ. Then $Q_{\xi} \equiv_{\mathcal{A}} \pi_{S[XY]}\sigma_{Y_{Q_{\xi_j}}=S[X]}(Q_{\xi_j} \times R(X,Y,Z))$, where $Y_{Q_{\xi_j}}$ is the set of attributes of the output relation of Q_{ξ_j}. By the induction hypothesis, there exists Q'_{ξ_j} strongly covered by $\mathcal{A}$ and $Q'_{\xi_j} \equiv Q_{\xi_j}$. Let Q_1 be $\pi_{S[XY]}\sigma_{Y_{Q'_{\xi_j}}=S[X]}Q'_{\xi_j} \times R(X,Y,Z)$, where $Y_{Q'_{\xi_j}}$ is the set of attributes of the output of Q'_{ξ_j}. Then $Q_1 \equiv_{\mathcal{A}} Q_{\xi}$. Similar to cases (4) and (3) above, one can show by induction on the structure of Q'_{ξ_j} that there is Q'_{ξ} such that $Q'_{\xi} \equiv_{\mathcal{A}} Q_1$ and Q'_{ξ} is strongly covered by $\mathcal{A}$.

Thus statement (1) holds, and so does Theorem 15(1).

Proof of Theorem 15(2). For any RA query Q that is covered by $\mathcal{A}$, by the definition, every max SPC sub-query Q_s of Q is covered by $\mathcal{A}$. Observe that the CQ translation of Q_s is also covered by $\mathcal{A}$ (see the proof of Theorem 4 in Chapter 2), By Lemma 8 of Chapter 2, Q_s has a boundedly evaluable query plan under $\mathcal{A}$. Therefore, Q has a boundedly evaluable query plan under $\mathcal{A}$, which is a composition of the bounded plans of all its max SPC sub-queries *w.r.t.* Q. □

3.2 Checking Covered RA Queries

We first give a constructive proof of Theorem 15(3) by providing an algorithm for checking covered queries, denoted by CovChk. Given an access schema $\mathcal{A}$ and an RA query Q, CovChk returns "yes" if Q is covered by $\mathcal{A}$, and "no" otherwise. Below we

```
Algorithm CovChk

Input: An RA query $Q$ and an access schema $\mathcal{A}$.
Output: "yes" if $Q$ is covered by $\mathcal{A}$ and "no" otherwise.

1.  identify the set $\mathcal{S}_Q$ of all max SPC sub-queries of $Q$
2.  for each max SPC sub-query $Q_s$ in $\mathcal{S}_Q$ do
3.      if $Q_s$ is not indexed under $\mathcal{A}$ then return "no";
4.      construct induced FDs $\Sigma_{Q_s,\mathcal{A}}$ for $Q_s$ and $\mathcal{A}$;
5.      if $\Sigma_{Q_s} \not\models \hat{X}_C^{Q_s} \to \hat{X}_{Q_s}$ then return "no";
6.  return "yes";
```

Figure 3.1: Algorithm CovChk

show a result stronger than Theorem 15(3).

Proposition 17: *Given an access schema $\mathcal{A}$ and an* RA *query Q, algorithm* CovChk *determines whether Q is covered by $\mathcal{A}$ in $O(|Q|^2 + |\mathcal{A}|)$ time.* □

Note that checking is conducted at the meta level on Q and $\mathcal{A}$ only, independent of (possibly big) datasets D.

The algorithm is shown in Fig. 3.1, consisting of two parts. It first finds the set $\mathcal{S}_Q$ of all max SPC sub-queries of Q (line 1). It then checks whether all queries in $\mathcal{S}_Q$ are covered by $\mathcal{A}$ (lines 2–5), and returns "yes" if so (line 6).

Identifying max SPC **sub-queries**. CovChk computes the set $\mathcal{S}_Q$ by a bottom-up scan of the query tree of Q. This is done in time linear in $|Q|$, since each relation of Q occurs in only one max SPC sub-query of Q, by the assumption that relation names in Q are distinct (see Section 3.1).

Checking coverage of SPC **sub-queries**. We next focus on how to check whether an SPC sub-query Q_s is covered by $\mathcal{A}$. The checking is based on its connection with the implication analysis of functional dependencies (FDs) [AHV95]. To establish the connection, we introduce the following notions.

Unification. A *unification function* ρ_U is an *attribute* renaming function: for all attributes A and A' in $\mathcal{S}_Q$, $\rho_U(A) = \rho_U(A')$ (assigned the same name) if and only if $\Sigma_Q \vdash A = A'$. For a set X of attributes, we denote by $\rho_U(X)$ the set $\{\rho_U(A) \mid A \in X\}$. Let $\mathcal{S}_Q$ also denote all the attributes in $\mathcal{S}_Q$; we refer to $\rho_U(\mathcal{S}_Q)$ as the *unified schema of Q*.

<u>*Induced* FDs</u>. For a relation name R that occurs in Q and a constraint $\phi = R(A \to B, N)$ in $\mathcal{A}$, we call $\rho_U(R[A]) \to \rho_U(R[B])$ an *induced* FD from Q and ϕ. We denote by $\Sigma_{Q,\mathcal{A}}$ the set of all induced FDs from Q and constraints in $\mathcal{A}$.

Example 7: For Q_1 and $\mathcal{A}_0$ of Example 5, define a unification function ρ_U such that $\rho_U(\text{friend}[\text{pid}])$ = pid, $\rho_U(\text{friend}[\text{fid}])$ = fid, $\rho_U(\text{dine}[\text{pid}])$ = fid, $\rho_U(\text{dine}[\text{cid}])$ = cid, $\rho_U(\text{dine}[\text{year}])$ = year, $\rho_U(\text{dine}[\text{month}])$ = month, $\rho_U(\text{cafe}[\text{cid}])$ = cid and $\rho_U(\text{cafe}[\text{city}])$ = city. Then $\Sigma_{Q_1,\mathcal{A}_0}$ consists of the following induced FDs: pid $\to$ fid, (fid, year, month) $\to$ cid, (fid, cid) $\to$ (fid, cid), and cid $\to$ city. □

We now give the connection between induced FDs and fetchable SPC queries. For an SPC query Q_s, let X_{Q_s} be the set of all its attributes that occur in its selection condition or projection attributes, and $X_C^{Q_s} \subseteq X_{Q_s}$ be the set of attributes A in Q_s such that $\Sigma_{Q_s} \vdash A = c$ for some constant c. Let $\hat{X}_{Q_s} = \rho_U(X_{Q_s})$ and $\hat{X}_C^{Q_s} = \rho_U(X_C^{Q_s})$. Then we have:

Lemma 18: *An* SPC *query Q_s is fetchable under $\mathcal{A}$ if and only if* $\Sigma_{Q_s,\mathcal{A}} \models \hat{X}_C^{Q_s} \to \hat{X}_{Q_s}$. □

Here $\Sigma \models \varphi$ denotes the standard FD implication: for all databases D, if D satisfies Σ, then D also satisfies φ (see [AHV95]).

Intuitively, $\hat{X}_C^{Q_s}$ is the set of attributes whose values are already provided by Q_s, and $\hat{X}_{Q_s}$ includes all the attributes whose values are needed for answering Q_s. The computation of $\text{cov}(Q,\mathcal{A})$ (Section 3.1 of Chapter 3) is a chasing process with $\mathcal{A}$ to deduce $\hat{X}_{Q_s}$ from $\hat{X}_C^{Q_s}$ (see [AHV95] for chasing). The process coincides precisely with the implication analysis of $\Sigma_{Q_s,\mathcal{A}} \models \hat{X}_C^{Q_s} \to \hat{X}_{Q_s}$. Indeed, $\text{cov}(Q,\mathcal{A})$ is deduced by "the transitivity" of the "FD part" $X \to Y$ in access constraints $R(X \to Y, N)$, where cardinality N is needed only for deciding the bounded size, not in the deduction of coverage. Formally, Lemma 18 can be verified by induction on the length of the chasing process.

Proof: Since $\Sigma_{Q_s,\mathcal{A}} \models \hat{X}_C^{Q_s} \to \hat{X}_{Q_s}$ iff $\hat{X}_{Q_s} \subseteq (\hat{X}_C^{Q_s})^*$ (cf. [AHV95]), to show that Q_s is fetchable via $\mathcal{A}$ iff $\Sigma_{Q_s,\mathcal{A}} \models \hat{X}_C^{Q_s} \to \hat{X}_{Q_s}$, we just need to show that $X_{Q_s} \subseteq \text{cov}(Q_s,\mathcal{A})$ iff $\hat{X}_{Q_s} \subseteq (\hat{X}_C^{Q_s})^*$, where $(\hat{X}_C^{Q_s})^*$ is the FD closure of $\hat{X}_C^{Q_s}$ under $\Sigma_{Q_s,\mathcal{A}}$. We prove this by showing the following:

(1) $X_{Q_s} \subseteq \text{cov}(Q_s,\mathcal{A})$ iff $\rho_U(X_{Q_s}) \subseteq \rho_U(\text{cov}(Q_s,\mathcal{A}))$; and

(2) $\rho_U(\text{cov}(Q_s,\mathcal{A})) = (\rho_U(X_C^{Q_s}))^*$ (recall $\rho_U(X) = \hat{X}$).

<u>*Proof of (1)*</u>. The $\Rightarrow$ direction is obvious. We show the $\Leftarrow$ direction, when $\rho_U(X_{Q_s}) \subseteq \rho_U(\text{cov}(Q_s,\mathcal{A}))$. Assume by contradiction that $\rho_U(X_{Q_s}) \subseteq \rho_U(\text{cov}(Q_s,\mathcal{A})$ but $X_{Q_s} \not\subseteq$

$\mathsf{cov}(Q_s,\mathcal{A})$. Then there exists an attribute A in X_{Q_s} such that $A \notin \mathsf{cov}(Q_s,\mathcal{A})$. Since $\rho_U(X_{Q_s}) \subseteq \rho_U(\mathsf{cov}(Q_s,\mathcal{A}))$, $\rho_U(A) \in \rho_U(\mathsf{cov}(Q_s,\mathcal{A}))$. By the definition of ρ_U, there must exist an attribute B of Q such that $\rho_U(A) = \rho_U(B)$ and $B \in \mathsf{cov}(Q_s,\mathcal{A})$. Thus, by the definition of $\mathsf{cov}(Q,\mathcal{A})$, $\Sigma_{Q_s} \vdash A = B$. Since $B \in \mathsf{cov}(Q_s,\mathcal{A})$, this means that $A \in \mathsf{cov}(Q_s,\mathcal{A})$, which contradicts the assumption.

Proof of (2). To show (2), we first define a chasing procedure that computes $\mathsf{cov}(Q_s,\mathcal{A})$ for any SPC query Q_s under $\mathcal{A}$. Based on it we then show $\rho_U(\mathsf{cov}(Q_s,\mathcal{A})) = (\rho_U(X_C^{Q_s}))^*$.

A chasing sequence of $\mathsf{cov}(Q_s,\mathcal{A})$ for Q_s is defined as

$$\mathsf{cov}(Q_s,\mathcal{A}) = \mathsf{cov}_0(Q_s,\mathcal{A}),\ldots,\mathsf{cov}_n(Q_s,\mathcal{A}),$$

such that (1) $\mathsf{cov}_0(Q_s,\mathcal{A}) = X_C^{Q_s}$, and (2) for each $i \geqslant 0$, $\mathsf{cov}_{i+1}(Q_s,\mathcal{A})$ is obtained by applying some rules given in the definition of coverage $\mathsf{cov}(Q,\mathcal{A})$ (Section 3.1) so that $\mathsf{cov}_i(Q_s,\mathcal{A}) \neq \mathsf{cov}_{i+1}(Q_s,\mathcal{A})$. Obviously such a chasing sequence is terminal; moreover, by the definition of $\mathsf{cov}(Q_s,\mathcal{A})$, the result $\mathsf{cov}_n(Q_s,\mathcal{A})$ of the chasing sequence (the last element) is exactly $\mathsf{cov}(Q_s,\mathcal{A})$ for Q_s and $\mathcal{A}$.

We next show $\rho_U(\mathsf{cov}(Q_s,\mathcal{A})) = (\rho_U(\mathsf{cov}_0(Q_s,\mathcal{A})))^*$ (thus $= (\rho_U(X_C^{Q_s}))^*$) by induction on the length n of the chase.

Basis. When $n = 1$, $\mathsf{cov}(Q_s,\mathcal{A}) = \mathsf{cov}_0(Q_s,\mathcal{A}) = X_C^{Q_s}$. Obviously $(X_C^{Q_s})^* = X_C^{Q_s}$ under $\Sigma_{Q_s,\mathcal{A}}$ in this case. Thus $\rho_U(\mathsf{cov}(Q_s,\mathcal{A})) = (\rho_U(\mathsf{cov}_0(Q_s,\mathcal{A})))^*$.

Induction step. Assume that for any such chasing sequence of length n with $n \leqslant k$ ($k \geqslant 1$), $\rho_U(\mathsf{cov}(Q_s,\mathcal{A})) = (\rho_U(X_C^{Q_s}))^*$. Consider the case when $n = k+1$, *i.e.*, $\mathsf{cov}(Q_s,\mathcal{A}) = \mathsf{cov}_0(Q_s,\mathcal{A}), \ldots, \mathsf{cov}_k(Q_s,\mathcal{A}), \mathsf{cov}_{k+1}(Q_s,\mathcal{A})$. Observe that $\mathsf{cov}(Q_s,\mathcal{A})$ also has a chasing $\mathsf{cov}_1(Q_s,\mathcal{A}),\ldots,\mathsf{cov}_{k+1}(Q_s,\mathcal{A})$, which is of length k. Thus by the induction hypothesis, $\rho_U(\mathsf{cov}(Q_s,\mathcal{A})) = (\rho_U(\mathsf{cov}_1(Q_s,\mathcal{A})))^*$. We next show that $(\rho_U(\mathsf{cov}_1(Q_s,\mathcal{A})))^* = (\rho_U(\mathsf{cov}_0(Q_s,\mathcal{A})))^*$ under $\Sigma_{Q_s,\mathcal{A}}$. Recall algorithm $\mathsf{cloFD}(X,\Sigma)$ for computing the closure X^* of X for a set Σ of FDs closures (Algorithm 8.2.7 in [AHV95]). One can easily verify that for each of the rules used for deducing $\mathsf{cov}_1(Q_s,\mathcal{A})$ from $\mathsf{cov}_0(Q_s,\mathcal{A})$, we have that $\mathsf{cloFD}(\mathsf{cov}_0(Q_s,\mathcal{A}),\Sigma_{Q_s,\mathcal{A}}) = \mathsf{cloFD}(\mathsf{cov}_1(Q_s,\mathcal{A}),\Sigma_{Q_s,\mathcal{A}})$. Thus, for any chasing sequence $\mathsf{cov}_0(Q_s,\mathcal{A}) = X_C^{Q_s}, \ldots, \mathsf{cov}_n(Q_s,\mathcal{A}) = \mathsf{cov}(Q,\mathcal{A})$, we have $\rho_U(\mathsf{cov}(Q_s,\mathcal{A})) = (\rho_u(\mathsf{cov}_0(Q_s,\mathcal{A})))^* = (\rho_U(X_C^{Q_s}))^*$. □

Lemma 18 reduces the problem of checking whether an SPC query Q_s is fetchable via $\mathcal{A}$ to the implication analysis of FDs. Based on the lemma, algorithm CovChk checks whether Q_s is fetchable by firstly constructing the set $\Sigma_{Q_s,\mathcal{A}}$ of induced FDs from Q_s and $\mathcal{A}$, and then checking whether $\Sigma_{Q_s,\mathcal{A}} \models \hat{X}_C^{Q_s} \rightarrow \hat{X}_{Q_s}$ by invoking a linear-time FD implication algorithm [AHV95] (lines 4–5). It checks whether Q_s is indexed under $\mathcal{A}$ simply by definition (line 3).

Example 8: Given Q_0 and $\mathcal{A}_0$ of Example 5, algorithm CovChk examines the max SPC sub-queries Q_1 and Q_2 of Q_0. It first computes the set $\Sigma_{Q_1,\mathcal{A}_0}$ of induced FDs from Q_1 and $\mathcal{A}_0$ (see Example 7), with $\hat{X}_{Q_1}$ = {pid, fid, cid, year, month, city} and $\hat{X}_C^{Q_1}$ = {pid, year, month, city}. It verifies that Q_1 is covered by $\mathcal{A}_0$ since $\Sigma_{Q_1,\mathcal{A}_0} \models \hat{X}_C^{Q_1} \rightarrow \hat{X}_{Q_1}$, and Q_1 is indexed by $\mathcal{A}$ (Example 6). Along the same lines, it finds that Q_2 is not covered, and concludes that Q_0 is not covered. In contrast, it finds that max SPC sub-queries Q_1 and Q_3 of Q'_0 are both covered by $\mathcal{A}_0$, and thus so is Q'_0. □

Correctness & Complexity. The correctness of CovChk follows from the definition of covered queries and Lemma 18. We next show that CovChk can be implemented in $O(|Q|^2 + |\mathcal{A}|)$ time. (1) It takes $O(|Q|)$ time to compute the set $\mathcal{S}_Q$ of all max SPC sub-queries of Q. (2) Checking whether all Q_s's in $\mathcal{S}_Q$ are indexed by $\mathcal{A}$ can be implemented in $O(|Q| + |\mathcal{A}|)$ time, by building a hash-index from relations in Q to associated constraints of $\mathcal{A}$ in $O(|Q| + |\mathcal{A}|)$ time before the iteration, such that it takes $O(|Q_s| + |\mathcal{A}_{Q_s}|)$ time to check whether Q_s is indexed by $\mathcal{A}$, where $\mathcal{A}_{Q_s}$ is the set of constraints in $\mathcal{A}$ that are defined on relations in Q_s. (3) It takes $O(|Q_s|^2)$ time to construct induced FDs $\Sigma_{Q_s,\mathcal{A}}$, and the size of $\Sigma_{Q_s,\mathcal{A}}$ is bounded by $|\mathcal{A}_{Q_s}|$ for each Q_s. (4) FD implication checking can be done in linear time (cf. [AHV95]), *i.e.*, $O(|\Sigma_{Q_s,\mathcal{A}}| + |X_C^{Q_s}| + |X_{Q_s}|) = O(|\mathcal{A}_{Q_s}| + |Q_s|)$ for each Q_s. Putting these together, CovChk is in $O(|Q|^2 + |\mathcal{A}|)$ time. This completes the proof of Proposition 17 and Theorem 15(3).

The notations of the paper are summarized in Table 3.1.

3.3 Generating Bounded Query Plans

We now verify Theorem 15(2) by proving a stronger result.

Theorem 19: *(1) For any* RA *query Q covered by an access schema $\mathcal{A}$, Q has a canonical bounded query plan under $\mathcal{A}$. (2) There exists an algorithm that given Q covered by $\mathcal{A}$, generates a canonical bounded query plan of length $O(|Q||\mathcal{A}|)$ in $O(|Q|(|Q| + |\mathcal{A}|))$ time.* □

Here canonical bounded query plans are boundedly evaluable query plans that capture covered RA queries. That is, every covered query Q warrants a boundedly evaluable query plan ξ. Better still, ξ can be generated in a bounded amount of time and has

Notation	Description
$\mathcal{A}$	(actualized) access schema
$\lvert\mathcal{A}\rvert$	the total length of access constraints in $\mathcal{A}$
$\lVert\mathcal{A}\rVert$	the number of constraints in $\mathcal{A}$
Q_s	a max SPC sub-query of Q
Σ_{Q_s}	equality $A = A'$ and $A = c$ derived from Q_s
X_Q	attributes in σ_C or π_Y of some max Q_s of Q
X_C^Q	attributes A such that $\Sigma_{Q_s} \vdash A = c$ for a Q_s of Q
X_Q^S	attributes in both S and X_{Q_s} for some Q_s of Q
$\rho_U(A)$	renaming of A with unification function ρ_U
$\rho_U(X)$	$\{\rho_U(A) \mid A \in X\}$
$\Sigma_{Q_s,\mathcal{A}}$	the set of induced FDs from Q_s and $\mathcal{A}$
ξ^c	canonical bounded query plan
$\xi_F^c(A)$	unit fetching plan for attribute A
$\mathcal{G}_{Q,\mathcal{A}}$	$\langle Q,\mathcal{A}\rangle$-hypergraph for Q and $\mathcal{A}$
Π_{V_S,u_A}	hyperpath from set V_S to node u_A

Table 3.1: Notations of Chapter 3

a bounded length, both determined by Q and $\mathcal{A}$, independent of the underlying datasets.

The proof is nontrivial. Below we first introduce canonical bounded query plans (Section 3.3.1). We then provide an algorithm with the property of Theorem 19(2) (Section 3.3.2).

3.3.1 Capturing Covered Queries with Query Plans

We introduce canonical query plans, and show that such plans characterize covered queries. That is, an RA query Q is covered by $\mathcal{A}$ *if and only if* Q has a canonical bounded query plan under $\mathcal{A}$. From this Theorem 19(1) follows.

Canonical bounded query plans. For an RA query Q under an access schema $\mathcal{A}$, a *canonical bounded query plan* ξ^c is a boundedly evaluable query plan for Q that consists of a *fetching plan* ξ_F^c, followed by an *indexing plan* ξ_I^c and then an *evaluation plan* ξ_E^c, defined as follows.

Fetching plan ξ_F^c: A fetching plan ξ_F^c is a sequence of *unit fetching plans* $\xi_F^c(A_1), \ldots,$ $\xi_F^c(A_m)$, for all attributes $A_1, \ldots, A_m$ in X_Q of Q, where $\xi_F^c(A_i)$ is inductively defined as follows (assuming A_i is in a max SPC sub-query Q_s of Q):

(i) if $A_i \in X_C^{Q_s}$, then $\xi_F^c(A_i)$ is $\{c\}$, where $\sigma_{A_i=c}$ is in Q_s;

(ii) if $\sigma_{A_i=A'}$ is in Q_s and there exists a unit fetching plan $\xi_F^c(A')$ for A', then $\xi_F^c(A_i)$ = $\xi_F^c(A')$; and

(iii) if there exists a constraint $R(W \to U, N)$ in $\mathcal{A}$ such that $A_i \in R[U]$, and moreover, if for each $w_i \in R[W] = \{w_1, \ldots, w_m\}$, there exists a unit fetching plan $\xi_F^c(w_i)$ for w_i, then $\xi_F^c(A_i)$ is:
   - ◦ $T_1 = \xi_F^c(w_1), \ldots, T_m = \xi_F^c(w_m)$,
   - ◦ $T_{m+1} = T_1 \times \cdots \times T_m$,
   - ◦ $T_{m+2} = \mathsf{fetch}(X \in T_{m+1}, R, U)$,
   - ◦ $T_{m+3} = \pi_{A_i}(T_{m+2})$.

That is, ξ_F^c fetches all necessary attribute values one by one, employing an access constraint of $\mathcal{A}$ in each step.

Indexing plan ξ_I^c. An indexing plan ξ_I^c is a sequence of *unit indexing plans* $\xi_I^c(S_1), \ldots, \xi_I^c(S_m)$ for all relations $S_1, \ldots, S_m$ in Q. For each S_i, let Q_s be the max SPC sub-query in which S_i occurs, $X_{Q_s}^{S_i} = \{A_1, \ldots, A_K\}$ be the set of attributes of S_i that also occur in X_{Q_s}, and $S_i(X \to Y, N)$ be a constraint in $\mathcal{A}$ that indexes S_i. Then $\xi_I^c(S_i)$ is as follows:

- ◦ $T_1 = \xi_F^c(A_1), \ldots, T_K = \xi_F^c(A_K)$,
- ◦ $T_{K+1} = T_1 \times \cdots \times T_K$,
- ◦ $T_{K+2} = \pi_{S_i[X]}(T_{K+1})$,
- ◦ $T_{K+3} = \mathsf{fetch}(X \in T_{K+2}, S_i, Y)$, and
- ◦ $T_{K+4} = T_{K+1} \cap T_{K+3}$ (expressed in terms of $\times$, σ, π).

That is, ξ_I^c ensures that each S_i in Q is indexed.

Evaluation plan ξ_E^c. Plan ξ_E^c is the RA expression of Q, in which each relation S_i in Q is replaced by T_k, where $T_k = \xi_I^c(S_i)$ is the output of the indexing plan $\xi_I^c(S_i)$ for S_i.

Intuitively, given a dataset D with $D \models \mathcal{A}$, a canonical bounded query plan ξ^c first executes ξ_F^c to fetch necessary data values from D via indices in $\mathcal{A}$. This is followed by ξ_I^c to combine and filter tuples for each relation that is needed for answering max SPC sub-queries of Q. Finally, ξ_E^c is executed against the fetched tuples instead of D directly. That is, ξ^c accesses data only via ξ_F^c and ξ_I^c.

By the definition of bounded evaluability, one can verify that a canonical bounded query plan is boundedly evaluable under $\mathcal{A}$. Moreover, canonical bounded plans characterize covered queries.

Lemma 20: *For* RA *query Q and access schema $\mathcal{A}$, (1) Q is fetchable via $\mathcal{A}$ iff Q has a fetching plan under $\mathcal{A}$; and (2) Q is indexed by $\mathcal{A}$ iff Q has an indexing plan under $\mathcal{A}$.* □

Proof: By the definition of indexed queries, Q is indexed by $\mathcal{A}$ iff Q has an indexing plan. Below we show that Q is fetchable via $\mathcal{A}$ iff Q has a fetching plan under $\mathcal{A}$.

$\boxed{\Rightarrow}$ Assume that Q is fetchable via $\mathcal{A}$. Then each max SPC sub-query Q_s is fetchable via $\mathcal{A}$, *i.e.,* $X_{Q_s} \subseteq \mathsf{cov}(Q_s, \mathcal{A})$. Thus for each attribute $A \in X_{Q_s}$, $A \in \mathsf{cov}(Q_s, \mathcal{A})$. Consider the chasing sequence $\mathsf{cov}(Q_s, \mathcal{A}) = \mathsf{cov}_0(Q_s, \mathcal{A}), \ldots, \mathsf{cov}_n(Q_s, \mathcal{A})$ described in the proof of Lemma 18, where $\mathsf{cov}_0(Q_s, \mathcal{A}) = X_C^{Q_s}$ and $\mathsf{cov}_n(Q_s, \mathcal{A}) = \mathsf{cov}(Q_s, \mathcal{A})$. There must exist $i \in [0,n]$ such that $A \in \mathsf{cov}_i(Q_s, \mathcal{A})$ but $A \notin \mathsf{cov}_{i-1}(Q_s, \mathcal{A})$ (if exists). We refer to $\mathsf{cov}_0(Q_s, \mathcal{A})$, ..., $\mathsf{cov}_i(Q_s, \mathcal{A})$ as the *deduced chasing* for attribute A. We next prove that there exists a unit fetching plan for A under $\mathcal{A}$ by induction on the length of the deduced chasing for A.

Basis. Assume that A has a deduced chasing of length 0. That is, $A \in X_C^{Q_s}$. Then a unit fetching plan for A is $\xi_F^c(A) = \{c\}$, where $\sigma_{A=c}$ is in Q_s.

Induction step. Suppose that whenever an attribute A in X_{Q_s} has a deduced chasing of length no longer than k, there exists a unit fetching plan $\xi_F^c(A)$ for A. Now consider the case when A has a deduced chasing $\mathsf{cov}_0(Q_s, \mathcal{A}), \ldots, \mathsf{cov}_k(Q_s, \mathcal{A})$ of length $k+1$. One can show that the statement holds on $k+1$ by examining all cases on which a rule is applied for deducing $\mathsf{cov}_k(Q_s, \mathcal{A})$ from $\mathsf{cov}_{k-1}(Q_s, \mathcal{A})$. As an example, we give details for one case here. Consider the case when the last rule is used, *i.e.,* if $R(X \to Y, N) \in \mathcal{A}$ and $R[X] \subseteq \mathsf{cov}(Q_s, \mathcal{A})$, then $R[Y] \subseteq \mathsf{cov}(Q_s, \mathcal{A})$. Then $\mathsf{cov}_0(Q_s, \mathcal{A}), \ldots, \mathsf{cov}_{k-1}(Q_s, \mathcal{A})$ form a deduced chasing of length k for each attribute A' in $R[X]$. By the induction hypothesis, there exists unit fetching plans $\xi_F^c(A')$ for each A' in $R[X]$. Then by the definition of unit fetching plan, there also exists a unit fetching plan for $\mathcal{A}$ under $\mathcal{A}$: $T_1 = \xi_F^c(A'_1), \ldots, T_m = \xi_F^c(A'_m)$ for $R[X] = \{A'_1, \ldots, A'_m\}$. $T_{m+1} = T_1 \times \cdots \times T_m$, $T_{m+2} = \mathsf{fetch}(X \in T_{m+1}, R, Y)$, $T_{m+3} = \pi_A(T_{m+2})$. The proof is similar for the other cases.

$\boxed{\Leftarrow}$ Assume that Q has a fetching plan under $\mathcal{A}$. Then for each attribute A in X_Q, *i.e.,* in X_{Q_s} of some max SPC sub-query Q_s of Q, there exists a fetching plan $\xi_F^c(A)$ for A under $\mathcal{A}$. We show that Q_s is fetchable via $\mathcal{A}$, *i.e.,* $X_{Q_s} \subseteq \mathsf{cov}(Q_s, \mathcal{A})$. It suffices to show $A \in \mathsf{cov}(Q_s, \mathcal{A})$. This can be readily verified by induction on the length of $\xi_F^c(A)$, by the definitions of unit fetching plans and fetchable queries. □

Theorem 19(1) follows from Lemma 20. Indeed, if Q is covered by $\mathcal{A}$, then Q has a fetching plan ξ_F^c and an indexing plan ξ_I^c under $\mathcal{A}$, which provide ξ_E^c with sufficient data values from D to compute $Q(D)$, for any $D \models \mathcal{A}$. These yield a canonical bounded query plan for Q under $\mathcal{A}$.

3.3.2 Generating Canonical Bounded Plans

We next give a constructive proof of Theorem 19(2) by developing an algorithm that, given an access schema $\mathcal{A}$ and an RA Q covered by $\mathcal{A}$, returns a canonical bounded query plan ξ^c of bounded length in $O(|Q|(|Q|+|\mathcal{A}|))$ time. The idea of the algorithm is to encode Q and $\mathcal{A}$ in a hypergraph representation such that (i) there is a certain hyperpath in the hypergraph *iff* Q is fetchable under $\mathcal{A}$; and (ii) each such hyperpath encodes a canonical fetching plan for Q under $\mathcal{A}$.

Below we first introduce structures used by the algorithm (see Table 3.1). We then present the algorithm.

$\langle Q, \mathcal{A}\rangle$-hypergraph $\mathcal{G}_{Q,\mathcal{A}}$. A *directed hypergraph $\mathcal{H}$* (cf. [AFF01]) is a pair (V, E), where V is a nonempty set of nodes and E is a set of hyperedges. A *hyperedge e* in E is an ordered pair (H, t), where $H \subseteq V$, $H \neq \emptyset$, and $t \in V \setminus H$. Here H and t are called the *head* and *tail* of e, and are denoted by $\mathsf{head}(e)$ and $\mathsf{tail}(e)$, respectively. The size $|\mathcal{H}|$ of $\mathcal{H}$ is the sum of the cardinality of its hyperedges, *i.e.,* $\sum_{e\in E} |\mathsf{head}(e)|$.

Given an RA query Q and an access schema $\mathcal{A}$, we use a hypergraph to encode the induced FDs for all max SPC sub-queries of Q. Let $\Sigma_{Q,\mathcal{A}}$ be the union of $\Sigma_{Q_s,\mathcal{A}}$ (the set of induced FDs for Q_s and $\mathcal{A}$) for all max SPC sub-queries Q_s in Q. We assume *w.l.o.g.* that for any two max SPC sub-queries Q_s and $Q_{s'}$ of Q, $\Sigma_{Q_s,\mathcal{A}} \cap \Sigma_{Q_{s'},\mathcal{A}} = \emptyset$. A *$\langle Q, \mathcal{A}\rangle$-hypergraph* (or simply hypergraph) $\mathcal{G}_{Q,\mathcal{A}}$ for Q and $\mathcal{A}$ is a directed hypergraph (V, E) derived from $\Sigma_{Q,\mathcal{A}}$ as follows.

(1) For each induced FD $X \rightarrow Y$ in $\Sigma_{Q,\mathcal{A}}$, with $X = \{x_1, \ldots, x_p\} (p \geqslant 1)$ and $Y \setminus X = \{y_1, \ldots, y_q\} (q \geqslant 1)$, there are $p+q+1$ nodes $u_{x_1}, \ldots, u_{x_p}, u_{y_1}, \ldots, u_{y_q}$ and u_Y in V to encode $x_1, \ldots, x_p, y_1, \ldots y_q$ and the set Y, respectively, and there exist $q+1$ hyperedges $e_1 = (\{u_{x_1}, \ldots, u_{x_p}\}, u_Y)$, $e_2 = (\{u_Y\}, u_{y_1})$, $\ldots$, $e_{q+1} = (\{u_Y\}, u_{y_q})$ in E.

(2) There is a dummy node r such that for each induced FD $\emptyset \rightarrow Y$ in $\Sigma_{Q,\mathcal{A}}$ with $Y = \{y_1, \ldots, y_q\}$ $(q \geqslant 1)$, there exist $q+1$ nodes u_Y, $u_{y_1}, \ldots, u_{y_q}$ in V and $q+1$ hyperedges $(\{r\}, u_Y)$, $(\{u_Y\}, u_{y_1})$, $\ldots$, $(\{u_Y\}, u_{y_q})$ in E.

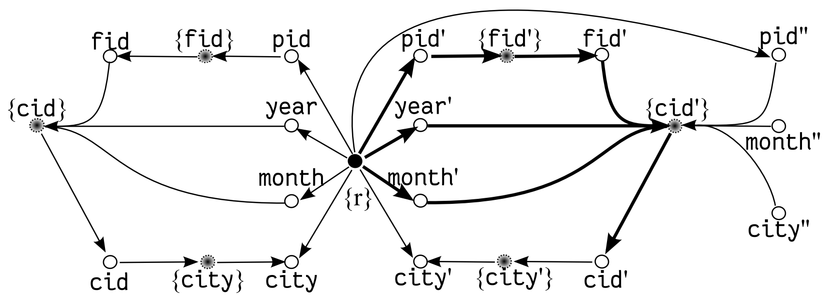


Figure 3.2: $\langle Q, \mathcal{A}\rangle$-hypergraph $\mathcal{G}_{Q_0', \mathcal{A}_0}$ for Q_0' and $\mathcal{A}_0$

(3) For each attribute A in $\hat{X}_C^Q = \{\rho_U(A) \mid A \in X_C^{Q_s}$, Q_s is a max SPC sub-query of $Q\}$, there exist a node u_A in V and a hyperedge $(\{r\}, u_A)$ in E.

Example 9: For Q_0' and $\mathcal{A}_0$ of Example 5, its $\langle Q, \mathcal{A}\rangle$-hypergraph $\mathcal{G}_{Q_0', \mathcal{A}_0}$ is depicted in Fig. 3.2, after the following conversions. We write $Q_0' = Q_1 - Q_3$ ($Q_3 = Q_1(\text{cid}) \bowtie_{\text{cid}=\text{cid}'} Q_2(\text{cid}')$) in the normal form of Section 3.1 such that it keeps relation names of Q_1 unchanged, and renames (a) each relation S in sub-query $Q_1(\text{cid})$ of Q_3 to S' (*e.g.,* dine of $Q_1(\text{cid})$ in Q_3 is renamed to dine$'$), and (b) each relation S in sub-query $Q_2(\text{cid}')$ of Q_3 to S'' (*e.g.,* dine of $Q_2(\text{cid})$ in Q_3 to dine$''$). In Fig. 3.2, we extend the unification function ρ_U given in Example 7. (a) For each attribute $S'[A]$ that occurs in sub-query $Q_1(\text{cid}')$ of Q_3, if $\rho_U(S[A]) = A$ in Q_1, then $\rho_U(S'[A]) = A'$ for $Q_1(\text{cid}')$ in Q_3. (b) For sub-query $Q_2(\text{cid})$ of Q_3, $\rho_U(\text{dine}''[\text{pid}]) = \text{pid}''$, $\rho_U(\text{dine}''[\text{cid}]) = \text{cid}'$, $\rho_U(\text{dine}''[\text{month}]) = \text{month}''$ and $\rho_U(\text{dine}''[\text{year}]) = \text{year}''$. □

Hyperpath. A *sub-hypergraph* of $\mathcal{H} = (V, E)$ is a hypergraph $\mathcal{H}' = (V', E')$ such that $V' \subseteq V$, $E' \subseteq E$, and E' is restricted to V'. A *hyperpath* [AFF01] in $\mathcal{H}$ from a set $S \subseteq V$ ($S \neq \emptyset$) of nodes to a target node $t \in V$ is a sub-hypergraph $\Pi_{S,t} = (V_{\Pi_{S,t}}, E_{\Pi_{S,t}})$ of $\mathcal{H}$ satisfying the following conditions: if $t \in S$, then $E_{\Pi_{S,t}} = \emptyset$; otherwise its $k \geqslant 1$ hyperedges can be ordered in a sequence $\langle e_1, \ldots, e_k\rangle$ such that (a) for any $e_i \in E_{\Pi_{S,t}}$, $\text{head}(e_i) \subseteq S \cup \{\text{tail}(e_1), \ldots, \text{tail}(e_{i-1})\}$; (b) $t = \text{tail}(e_k)$; and (c) no sub-hypergraph of $\Pi_{S,t}$ other than itself is a hyperpath from S to t in $\mathcal{H}$.

For example, a hyperpath $\Pi_{\{r\}, u_{\text{cid}'}}$ from r to $u_{\text{cid}'}$ in $\mathcal{G}_{Q_0', \mathcal{A}_0}$ of Example 9 is highlighted in bold in Fig. 3.2.

We now establish the connection between hyperpaths and canonical fetching plans as follows.

Lemma 21: *For any* RA *query Q, access schema $\mathcal{A}$ and attribute A in X_Q of Q, there exists a unit fetching plan $\xi^c(A)$ for Q under $\mathcal{A}$* if and only if *there exists a hyperpath* *from r to* $u_{\rho_U(A)}$ *in the hypergraph* $\mathcal{G}_{Q,\mathcal{A}}$. □

Proof: This is verified by giving translation algorithms Γ_ξ from $\xi^c(A)$ to $(\{r\}, u_{\rho_U(A)})$, and Γ_r from $(\{r\}, u_{\rho_U(A)})$ to $\xi^c(A)$.

$\boxed{\Rightarrow}$ Below we outline Γ_r, which will be used later in our algorithm. Given a hyperpath $\Pi_{\{r\},u_A}$ from $\{r\}$ to u_A, Γ_r inductively generates fetching plans as follows: (a) if $\Pi_{\{r\},u_A}$ is a hyperedge $(\{r\}, u_A)$ constructed in case (3) of $\langle Q, \mathcal{A}\rangle$-hypergraph above, then return $T_1 = \{c\}$; (b) if $\Pi_{\{r\},u_A}$ is a hyperedge $(\{r\}, u_Y)$ constructed in case (2) for induced FD $\emptyset \to Y$, then return $T_1 = \xi^c_F(A')$; and (c) if the last hyperedge of $\Pi_{\{r\},u_A}$ is a hyperedge (V_Y, u_A) constructed in case (1) of $\langle Q, \mathcal{A}\rangle$-hypergraph, and if for each u_{B_i} in $V_S = \{u_{Y_1}$, $\ldots$, $u_{Y_p}\}$, the unit fetching plan translated from hyperpath $\Pi_{\{r\},u_Y}$ is $\xi^c_F(Y_i)$, then return $T_1 = \xi^c_F(Y_1), \ldots, T_p = \xi^c_F(Y_p)$, $T_{p+1} = T_1 \times \cdots \times T_p$, $T_{p+2} = \mathsf{fetch}(X \in T_{p+1}, R, Y)$, and $T_{p+3} = \pi_A(T_{p+3})$.

$\boxed{\Leftarrow}$ We prove it by presenting an algorithm Γ_ξ that translates a unit fetching plan $\xi^c_F(A)$ to a hyperpath $(\{r\}, u_{\rho_U(A)})$. Given $\xi^c_F(A)$, algorithm Γ_ξ inductively generates a hyperpath $(\{r\}$, $u_{\rho_U(A))})$ as follows. (a) If $\xi^c_F(A)$ is of case (i) in the definition of unit fetching plans (Section 3.3.1), *i.e.,* $T_1 = \{c\}$, then return hyperedge $(\{r\}, u_{\rho_U(A)})$ directly. (b) If $\xi^c_F(A)$ is of case (ii), *i.e.,* $\xi^c_F(A) = \xi^c_F(A')$, then return hyperpath $(\{r\}, u_{\rho_U(A')})$. (c) If $\xi^c_F(A)$ is derived from case (iii), *i.e.,* there exists $R(W \to U, N)$ in $\mathcal{A}$ such that $A \in R[U]$ and for each $A'_i \in R[W] = \{A'_1, \ldots, A'_m\}$, there exists a unit fetching plan $\xi^c_F(A'_i)$ for A_i. Then Γ_ξ first finds a sub-hypergraph $\mathcal{G}_{\{r\},u_{\rho_U(A)}}$ in $\mathcal{G}_{Q,\mathcal{A}}$ that connects $\{r\}$ to $u_{\rho_U(A)}$. This ensures that there exists a hyperpath from $\{r\}$ to $u_{\rho_U(A)}$ in $\mathcal{G}_{\{r\},u_{\rho_U(A)}}$ (and thus in $\mathcal{G}_{Q,\mathcal{A}}$). Here $\mathcal{G}_{\{r\},u_{\rho_U(A)}}$ consists of the following: (i) the last two hyperedges in $\mathcal{G}_{\{r\},u_{\rho_U(A)}}$ are $(\{u_{\rho_U(A'_1)}, \ldots, u_{\rho_U(A'_m)}\}, u_{\rho_U(R[U])})$ and $(\{u_{\rho_U(R[U])}\}, u_{\rho_U(A)})$; and (ii) there are n hyperpaths $\Pi_{\{r\},u_{\rho_U(A'_i}}$ in $\mathcal{G}_{\{r\},u_{\rho_U(A)}}$ for each A'_i. Algorithm Γ_ξ then returns the hyperpath from $\{r\}$ to $u_{\rho_U(A)}$. One can readily verify that the algorithm returns a hyperpath $\Pi_{\{r\},u_{\rho_U(A)}}$ whenever there exists a unit fetching plan for A under $\mathcal{A}$. □

Lemma 21 tells us that to get a canonical fetching plan for Q under $\mathcal{A}$, it suffices to find hyperpaths from r to u_A in $\mathcal{G}_{Q,\mathcal{A}}$ for all attribute $A \in X_Q$. Based on this we develop our algorithm for canonical bounded plan generation.

Algorithm. The algorithm, denoted by QPlan and shown in Fig. 3.3, takes as input an access schema $\mathcal{A}$ and an RA query Q covered by $\mathcal{A}$; it returns a canonical bounded query plan $\mathcal{P}_{Q,\mathcal{A}}$ for Q under $\mathcal{A}$. It generates $\mathcal{P}_{Q,\mathcal{A}}$ in three steps: it first generates unit

```
Algorithm QPlan

Input: An access schema $\mathcal{A}$ and an RA query $Q$ covered by $\mathcal{A}$.
Output: A canonical bounded query plan $\xi^c$ for $Q$ under $\mathcal{A}$.

1.  construct the $\langle Q, \mathcal{A}\rangle$-hypergraph $\mathcal{G}_{Q,\mathcal{A}}$ for $Q$ and $\mathcal{A}$;
2.  $L_F[\,] := nil$; $\mathcal{P}_{Q,\mathcal{A}} = nil$;
3.  for each attribute $A \in X_Q$ of $Q$ do
      /* find a hyperpath from $r$ to $u_{\rho_U(A)}$ in $\mathcal{G}_{Q,\mathcal{A}}$ for $\rho_U(A)$ */
4.    $\Pi_{\{r\},u_{\hat{A}}} :=$ findHP$(r, u_{\hat{A}}, \mathcal{G}_{Q,\mathcal{A}})$; /* $\hat{A} = \rho_U(A)$ */
      /* translate hyperpath to a unit fetching plan for $A$ */
5.    $L_F[A] :=$ transQP$(\Pi_{\{r\},u_{\hat{A}}})$;
6.    append $L_F[A]$ to $\mathcal{P}_{Q,\mathcal{A}}$;
7.  for each relation $S$ in $Q$ do
8.    construct indexing plan $\xi^c_I(S)$ with $L_F[A]$ for all $A$ in $X^S_Q$;
9.    append $\xi^c_I(S)$ to $\mathcal{P}_{Q,\mathcal{A}}$;
10. append evaluation plan $\xi^c_E$ for $Q$ to $\mathcal{P}_{Q,\mathcal{A}}$;
11. return $\mathcal{P}_{Q,\mathcal{A}}$;
```

Figure 3.3: Algorithm QPlan

fetching plans for attributes in X_Q (lines 1-6). It then builds an indexing plan $\xi^c_I(S)$ for each relation name S that occurs in Q on top of the fetching plans (lines 7-9). Finally it adds the evaluation plan ξ^c_E (line 10).

More specifically, it first constructs the $\langle Q, \mathcal{A}\rangle$-hypergraph $\mathcal{G}_{Q,\mathcal{A}}$ for Q and $\mathcal{A}$ (line 1), and initializes data structures for storing unit fetching plans (L_F) and the final query plan ($\mathcal{P}_{Q,\mathcal{A}}$) (line 2). It then iteratively finds unit fetching plans for attributes in X_Q (lines 3-6). For each attribute A in X_Q, it finds a hyperpath $\Pi_{\{r\},u_A}$ from r to u_A that encodes A in $\mathcal{G}_{Q,\mathcal{A}}$, by invoking procedure findHP (line 4; not shown). Here findHP can be implemented in $O(|\mathcal{G}_{Q,\mathcal{A}}|) = O(|Q| + |\mathcal{A}|)$ time by traversing $\mathcal{G}_{Q,\mathcal{A}}$ [AFF01]. It then converts the hyperpath into a unit fetching plan $\xi^c_F(A)$ via procedure transQP (line 5), which is the translation algorithm Γ_r described in the proof sketch of Lemma 21; it adds the plan to $\mathcal{P}_{Q,\mathcal{A}}$ (line 6). After these, algorithm QPlan generates indexing plans for all relations in Q, by manipulating the unit fetching plans stored in L_F, following the definition of indexing plan (lines 7-9). It finally adds the evaluation plan of Q to $\mathcal{P}_{Q,\mathcal{A}}$ (line 10), and returns $\mathcal{P}_{Q,\mathcal{A}}$ (line 11).

Example 10: Given Q_0' and $\mathcal{A}_0$ of Example 5, algorithm QPlan first constructs the hypergraph $\mathcal{G}_{Q_0',\mathcal{A}_0}$ shown in Fig 3.2 for Q_0' and $\mathcal{A}_0$. It then iteratively finds unit fetching plans for attributes in $X_{Q_0'}$. Take cid' in sub-query Q_1 of Q_0' as an example (recall the setting from Example 9). It finds a hyperpath $\Pi_{\{r\},u_{\mathsf{cid}'}}$ in $\mathcal{G}_{Q_0',\mathcal{A}_0}$, marked in bold in Fig. 3.2. It then translates $\Pi_{\{r\},u_{\mathsf{cid}'}}$ into a unit fetching plan consisting of T_1-T_9 given in Example 5. Similarly, it generates unit fetching plans for all the other attributes in $X_{Q_0'}$. It then finds indexing plans for relations in Q_0'. For instance, an indexing plan for dine'' is as follows: $T_1' = T_9$ (since $T_1, \ldots, T_9$ form a unit fetching plan for cid'), $T_2' = \{p_0\}$, $T_3' = T_1' \times T_2'$, $T_4' = \mathsf{fetch}(X \in T_3', \mathsf{dine}'', (\mathsf{pid}'', \mathsf{cid}'))$. Finally, it adds the evaluation plan. Below is a complete canonical bounded query plan for Q_0' under $\mathcal{A}_0$, which is essentially the same as the one given in Example 5, if we overlook changes introduced by normalizing Q_0 and actualizing access schema $\mathcal{A}_0$ described in Section 3.1.

$T_1 = \{p_0\}$ $(\xi_F^c(\mathsf{pid}))$, $T_2 = \mathsf{fetch}(X \in T_1, \mathsf{friend}, \mathsf{fid})$;
$T_3 = \pi_{\mathsf{fid}}(T_2)$ $(\xi_F^c(\mathsf{fid})$; $T_4 = \{2015\}$ $(\xi_F^c(\mathsf{year}))$;
$T_5 = \{\text{MAY}\}$ $(\xi_F^c(\mathsf{month})$; $T_6 = T_3 \times T_4$; $T_7 = T_6 \times T_5$;
$T_8 = \mathsf{fetch}(X \in T_7, \mathsf{dine}, \mathsf{cid})$; $T_9 = \pi_{\mathsf{cid}}(T_8)$ $(\xi_F^c(\mathsf{cid}))$;
$T_{10} = \{\text{NYC}\}$ $(\xi_F^c(\mathsf{city}))$;
$T_{11}, \ldots, T_{20}$; $T_{21} = \{p_0\}$ $(\xi_F^c(\mathsf{pid}''))$.

Here T_{11} - T_{20} are unit fetching plans for attributes in Q_3, and are the same as T_1 - T_{10} *w.r.t.* attribute renaming.

An indexing plan $\xi_I^c(\mathsf{dine}'')$ for relation dine'' is: $T_1^I = T_{19}$; $T_2^I = T_{20}$; $T_3^I = T_{19} \times T_{20}$; $T_4^I = \mathsf{fetch}(X \in T_3^I, \mathsf{dine}, (\mathsf{pid}, \mathsf{cid}))$; similar for other relations. Finally, an evaluation plan for Q_0' under $\mathcal{A}_0$ is exactly Q_0' with each relation name S replaced by the T_S with $T_S = \xi_I^c(S)$. □

Correctness & Complexity. The correctness of QPlan is ensured by Theorem 19(1) and Lemma 21. By Theorem 19(1), a covered Q has a canonical bounded query plan, including a unit canonical fetching plan $\xi_F^c(A)$ for each attribute A in X_Q of Q. By Lemma 21, there exists a hyperpath from r to u_A for each $A \in X_Q$, encoding $\xi_F^c(A)$. Hence QPlan is warranted to be able to find such a plan.

Algorithm QPlan can be implemented in $O(|Q|(|Q| + |\mathcal{A}|))$ time. Indeed, (1) constructing the $\langle Q, \mathcal{A} \rangle$-hypergraph $G_{Q,\mathcal{A}}$ takes $O(|Q| + |\mathcal{A}|)$ time. (2) In each iteration (lines 3-6), findHP takes $O(|Q| + |\mathcal{A}|)$ time to find hyperpath $\Pi_{\{r\},u_{\hat{A}}}$ (cf. [AFF01]), and transQP takes $O(|\Pi_{\{r\},u_{\hat{A}}}|) = O(\|\mathcal{A}\|)$ time to translate P into a unit fetching plan, where $\|\mathcal{A}\|$ denotes the cardinality of $\mathcal{A}$. There are no more than $|Q|$ iterations. (3) Indexing plan generation takes $O(|Q|)$ time in total. (4) The size of the evaluation plan is bounded by $|Q|$. Putting these together, QPlan takes $O(|Q|+|\mathcal{A}|)$ + $O(|Q|(|Q|+|\mathcal{A}|+\|\mathcal{A}\|))$ + $O(|Q|)$ + $O(|Q|)$ = $O(|Q|(|Q|+|\mathcal{A}|))$ time in total.

The lemma below completes the proof of Theorem 19(2).

Lemma 22: *Given an* RA *query Q covered by $\mathcal{A}$,* QPlan *finds a canonical bounded query plan of length $O(|Q||\mathcal{A}|)$.* □

Proof: Observe the following: (i) the length of a unit fetching plan Q under $\mathcal{A}$ is bounded by $O(|\mathcal{A}|)$; (ii) there are at most $|Q|$ unit fetching plans; (iii) the length of an indexing plan for a relation S of Q under $\mathcal{A}$ is bounded by $O(|Q|)$; and (iv) the length of an evaluation plan for Q under $\mathcal{A}$ is bounded by $O(|Q|)$. Hence, the total length of a canonical bounded plan for Q under $\mathcal{A}$ is in $O(|Q||\mathcal{A}|)$. □

3.4 Optimization Problem for Covered Queries

In this section, we study an optimization problem for bounded evaluability, referred to as the *access minimization problem* and denoted by $\mathsf{AMP}(Q,\mathcal{A})$. It is stated as follows.

- Input: Access schema $\mathcal{A}$, RA query Q covered by $\mathcal{A}$.
- Output: A subset $\mathcal{A}_m \subseteq \mathcal{A}$ such that Q is also covered by $\mathcal{A}_m$ and $\mathcal{A}_m$ is *minimum*, *i.e.,* for any other subset $\mathcal{A}' \subseteq \mathcal{A}$, if Q is also covered by $\mathcal{A}'$, then $\sum_{R(X\to Y,N)\in\mathcal{A}_m} N \leqslant \sum_{R(X\to Y,N)\in\mathcal{A}'} N$.

That is, it is to identify a small set $\mathcal{A}_m$ of access constraints in $\mathcal{A}$ that covers Q and moreover, $\mathcal{A}_m$ estimates a "minimum" amount of data to be accessed for answering Q. It also suggests how many access constraints we need to cover a query, and the size of indices built for the constraints.

While useful, the problem is hard. We show that the problem is intractable (Section 3.4.1). Nonetheless, we provide efficient algorithms with performance guarantees (Section 3.4.2).

3.4.1 Intractability and Approximation Hardness

The decision version of AMP, denoted $\mathsf{dAMP}(Q,\mathcal{A},K)$, is to decide, given access schema $\mathcal{A}$, RA query Q covered by $\mathcal{A}$ and a natural number K, whether there exists $\mathcal{A}_m \subseteq \mathcal{A}$ such that A_m covers Q and $\sum_{R(X\to Y,N)\in\mathcal{A}_m} N \leqslant K$. Its corresponding optimization problem, denoted by $\mathsf{oAMP}(Q, \mathcal{A})$, is to find the minimum K for $\mathsf{dAMP}(Q, \mathcal{A}, K)$ to answer "yes".

We also study two practical special cases. We say that a $\langle Q, \mathcal{A}\rangle$-hypergraph $\mathcal{G}_{Q,\mathcal{A}}$ is *acyclic* if the graph representation $G_{Q,\mathcal{A}}$ of $\mathcal{G}_{Q,\mathcal{A}}$ is acyclic, where $G_{Q,\mathcal{A}}$ is a directed graph derived from $\mathcal{G}_{Q,\mathcal{A}}$ by replacing each hyperedge $e = (\{u_1, \ldots, u_p\}, v)$ with p edges (u_1, v), ..., (u_p, v). Intuitively, $\mathcal{G}_{Q,\mathcal{A}}$ is acyclic when the dependency relation on attributes of Q imposed by $\mathcal{A}$ is not "recursive". We study the following two special cases of $(Q, \mathcal{A})$:

- *acyclic*: when $\mathcal{G}_{Q,\mathcal{A}}$ is acyclic; and
- *elementary*: for each $\phi = R(X \to Y, N)$ in $\mathcal{A}$, either ϕ is an indexing constraint (*i.e.,* $X = Y$, see Section 3.1), or a *unit* constraint, *i.e.,* when $|X| = |Y| = 1$.

Both cases are quite common in practice: access constraints rarely incur recursive dependencies, and are often of the form of indexing or unit constraints. For example, (1) Q'_0 and $\mathcal{A}_0$ in Example 5 are an acyclic case since $\mathcal{G}_{Q'_0, A_0}$ (Fig. 3.2) is acyclic; and (2) Q'_0 and $\mathcal{A}_0 \setminus \{\psi_2\}$ are an elementary case.

These problems are nontrivial, even their special cases.

Theorem 23: *(1)* $\mathsf{dAMP}(Q, \mathcal{A}, K)$ *is* NP*-complete.*
(2) $\mathsf{oAMP}(Q, \mathcal{A})$ *is not approximable within* $c * \log |X_Q \setminus X_C^Q|$ *for any constant* $c > 0$.
(3) When $(Q, \mathcal{A})$ *is acyclic or elementary,* $\mathsf{dAMP}(Q, \mathcal{A}, K)$ *remains* NP*-hard, and* $\mathsf{oAMP}(Q, \mathcal{A})$ *is not in* APX. □

The class APX is the set of NP optimization problems that can be approximated by a constant-factor approximation algorithm, *i.e.,* a PTIME algorithm within *some* constant.

Proof: (1) We first show the $\mathsf{dAMP}(Q, \mathcal{A}, K)$ is NP-complete.

<u>Upper bound</u>. We show that $\mathsf{dAMP}(Q, \mathcal{A}, K)$ is in NP by giving an NP algorithm. It works as follows.

- Guess a subset $\mathcal{A}'$ of $\mathcal{A}$ with $\sum_{R(X \to Y, N) \in \mathcal{A}'} N \leqslant K$.
- Check whether Q is covered by $\mathcal{A}'$. If so, return "yes".

The algorithm is in NP since checking is doable in PTIME.

<u>Lower bound</u>. We show that $\mathsf{dAMP}(Q, \mathcal{A}, K)$ is NP-hard by reduction from the MINIMUM SET COVER problem, denoted by MSC. Given a collection C of subsets of a finite set S and a natural number k, MSC is to decide whether there exists a cover of C, *i.e.,* a subset $C' \subseteq C$ such that $|C'| \leqslant k$ and $\bigcup_{c \in C'} c = S$. It is known that MSC is NP-complete

(cf. [Pap94]), when each subset in C has 3 elements.

Given an instance of MSC, *i.e.,* C, S and k, we construct a database schema $\mathcal{R}$, an access schema $\mathcal{A}$ over $\mathcal{R}$, an RA Q over $\mathcal{R}$ that is covered by $\mathcal{A}$, and a natural number k'. We show that there exists $\mathcal{A}' \subseteq \mathcal{A}$ such that $\sum_{R(X \to Y, N) \in \mathcal{A}'} N \leqslant k'$ and Q is covered by $\mathcal{A}'$ if and only if there exists a cover $C' \subseteq C$ for S with $|C'| \leqslant k$.

We give the reduction as follows. (a) The relational schema $\mathcal{R}$ consists of n relations $R_i(A, B_1, B_2, B_3, E)$, where $n = |C|$, the number of subsets in C. For each $i \in [1, n]$, $R_i(A, B_1, B_2, B_3)$ encodes subset C_i in C. (b) The access schema $\mathcal{A}$ consists of $2n$ constraints. For each $i \in [1, n]$, 2 constraints are defined: $R_i(A \to (B_1, B_2, B_3), 1)$, $R_i((A, B_1, B_2, B_3, E) \to (A, B_1, B_2, B_3, E), 1)$. The first one indicates that given an A-value denoting subset C_i, the elements of C_i are fetchable. The second one ensures that relation R_i is indexed by $\mathcal{A}$. (c) We let $k' = k + n$. (d) We define an RA query Q (in fact in SPC) as follows:

$$\pi_Y \sigma_F (R_1(A, B_1, B_2, B_3, E) \times \cdots \times R_n(A, B_1, B_2, B_3, E)),$$

where (i) $Y = \{R_1[A], R_1[B_1], R_1[B_2], R_1[B_3], R_1[E], \ldots, R_n[A], R_n[B_1], R_n[B_2], R_n[B_3], R_n[E]\}$, and (ii) the selection condition $F = F_1 \wedge F_2$ is defined as follows:

- F_1 is a conjunction of $R_i[B_p] = R_j[B_q]$, $i, j \in [1, n]$ and $p, q \in [1, 3]$, in which $R_i[B_p] = R_j[B_q]$ is in F iff for $C_i = \{X_i^1, X_i^2, X_i^3\}$ and $C_j = \{X_j^1, X_j^2, X_j^3\}$, X_i^p and X_j^q are the same element in S in the instance of MSC (assume *w.l.o.g.* that elements in each C_i are ordered);
- $F_2 = R_1[A] = 1 \wedge \ldots \wedge R_n[A] = 1 \wedge R_1[E] = 1 \wedge \ldots \wedge R_n[E] = 1$.

Intuitively, F_2 is to set X_C^Q to be $\{R_i[A], R_i[E] \mid i \in [1, n]\}$ and F_1 is to ensure that R_i of Q encodes C_i of C.

Observe that Q is covered by $\mathcal{A}$ already. We next show that there exists a cover $C' \subseteq C$ of S with $|C'| \leqslant k$ if and only if there exists $\mathcal{A}_m \subseteq \mathcal{A}$ such that $\sum_{R(X \to Y, N) \in \mathcal{A}_m} N \leqslant k'$ and Q is covered by $\mathcal{A}_m$.

$\boxed{\Rightarrow}$ First assume that there exists a cover C' of S with $|C'| \leqslant k$. Define $\mathcal{A}_m$ as follows: for each $i \in [1, n]$, if $C_i \in C'$, then $\mathcal{A}_m$ includes 2 constraints $R_i(A \to (B_1, B_2, B_3), 1)$ and $R_i((A, B_1, B_2, B_3, E) \to (A, B_1, B_2, B_3, E), 1)$; otherwise we take only $R_i((A, B_1, B_2, B_3, E) \to (A, B_1, B_2, B_3, E), 1)$ in $\mathcal{A}_m$. Observe that Q is indexed by $\mathcal{A}_m$. Assume by contradiction that Q is not fetchable via $\mathcal{A}_m$. Then there exist $i \in [1, n]$ and $p \in [1, 3]$ such that $R_i[B_p] \notin \mathsf{cov}(Q, \mathcal{A}_m)$. Since C' is a cover of S, there exists $C_i \in C'$ such that element $e_{ip} \in S$ that encodes $R_i[B_p]$ is contained by C_i.

Therefore, $R_i(A \to (B_1, B_2, B_3), 1)$ is in $\mathcal{A}_m$ by the definition of $\mathcal{A}_m$. This means that $R_i[B_p] \in \mathsf{cov}(Q, \mathcal{A}_m)$ by the definition of $\mathsf{cov}(Q, \mathcal{A}_m)$, a contradiction to the assumption. Therefore, Q is also fetchable via $\mathcal{A}_m$. That is, there exists $\mathcal{A}_m \subseteq \mathcal{A}$ such that $|\mathcal{A}_m| \leqslant k'$ and Q is covered by $\mathcal{A}_m$.

$\boxed{\Leftarrow}$ Conversely, assume that there exists $\mathcal{A}_m \subseteq \mathcal{A}$ such that $|\mathcal{A}_m| \leqslant k' = k + n$ and Q is covered by $\mathcal{A}_m$. By the definitions of Q and $\mathcal{A}$, $\mathcal{A}_m$ must contain $R_i((A, B_1, B_2, B_3, E) \to (A, B_1, B_2, B_3, E), 1)$ for each $i \in [1, n]$. So $\mathcal{A}_m$ contains another k access constraints of form $R_i(A \to (B_1, B_2, B_3), 1)$. Let C' consist of the following: for each $R_i(A \to (B_1, B_2, B_3), 1)$ in $\mathcal{A}_m$, we include C_i in C'. Then one can readily verify that C' is a cover of S along the same line as the proof of Lemma 18 given above.

(2) To show that $\mathsf{AMP}(Q, \mathcal{A})$ cannot be approximated within $c \log |X_Q \setminus X_C^Q|$, we prove the following. Let $\mathsf{AMP}'(Q, \mathcal{A})$ be the same as $\mathsf{AMP}(Q, \mathcal{A})$ except that the object function is $\sum_{R(X \to Y, N) \in \mathcal{A}'_{ni}} N$ for any feasible solution $\mathcal{A}' \subseteq \mathcal{A}$, where $\mathcal{A}'_{ni} \subseteq \mathcal{A}'$ consists of constraints in $\mathcal{A}'$ that are not indexing constraints. Similarly, the optimization problem $\mathsf{oAMP}'(Q, \mathcal{A})$ is defined. We prove the following lemmas.

Lemma 24: $\mathsf{oAMP}'(Q, \mathcal{A})$ *is approximable within* $2f(Q, \mathcal{A})$ *if* $\mathsf{oAMP}(Q, \mathcal{A})$ *is approximable within* $f(Q, \mathcal{A})$. □

Lemma 25: *For constant c,* $\mathsf{oAMP}'(Q, \mathcal{A})$ *is not approximable within* $c \log |X_Q \setminus X_C^Q|$. □

Theorem 23(2) immediately follows from Lemmas 24 and 25. Below we first prove Lemma 24 and then Lemma 25.

Proof of Lemma 24. Suppose that $\mathsf{oAMP}(Q, \mathcal{A})$ has a PTIME approximation algorithm Γ with approximation bound $f(Q, \mathcal{A})$. Below we show that Γ is also a $f(Q, \mathcal{A})$-approximation algorithm for $\mathsf{oAMP}'(Q, \mathcal{A})$. Let $\mathcal{A}_Q$ be the output of $\Gamma(Q, \mathcal{A})$, and $c(\cdot)$ and $c'(\cdot)$ be the objective functions of oAMP and oAMP$'$, respectively. Let $\mathcal{A}'_m$ and $\mathcal{A}_m$ be the optimum solution to $\mathsf{oAMP}'(Q, \mathcal{A})$ and $\mathsf{oAMP}(Q, \mathcal{A})$, respectively. Then $\frac{c'(\mathcal{A}_Q)}{c'(\mathcal{A}'_m)} \leqslant \frac{c(\mathcal{A}_Q)}{c(\mathcal{A}'_m)} \leqslant \frac{c(\mathcal{A}_Q)}{c(\mathcal{A}_m) - \|Q\|} \leqslant \frac{c(\mathcal{A}_Q)}{c(\mathcal{A}_m) - c(\mathcal{A}_m)/2} = 2f(Q, \mathcal{A})$, where $\|Q\|$ denotes the number of relations that occur in Q. From this Lemma 24 follows. □

Proof of Lemma 25. To show Lemma 25, we give an L-reduction from MSC (optimization version) to $\mathsf{oAMP}'(Q, \mathcal{A})$, *i.e.*, a pair of functions $f : I_{\mathsf{MSC}} \to I_{\mathsf{oAMP}'}$, $g : I_{\mathsf{MSC}} \times SOL_{\mathsf{oAMP}'}(f(I_{\mathsf{MSC}})) \to SOL_{\mathsf{MSC}}(I_{\mathsf{MSC}})$, where I_P is the set of all instances of problem

P, and $SQL_P(x)$ is the set of all feasible solutions to an instance x of problem P, such that there are constants α and β satisfying the following: conditions: (i) $OPT_{\mathsf{oAMP}'}(f(x)) \leqslant \alpha OPT_{\mathsf{MSC}}(x)$ for any $x \in I_{\mathsf{MSC}}$, where $OPT_P(x)$ is the optimum solution to instance x of P; and (ii) $|OPT_{\mathsf{MSC}}(x) - c_{\mathsf{MSC}}(g(f(x),s))| \leqslant \beta |OPT_{\mathsf{oAMP}'}(f(x)) - c_{\mathsf{oAMP}'}(s)|$, for each $s \in SOL_{\mathsf{oAMP}'}(f(x))$.

We define f, g, α, β as follows. (a) Function f maps instances of MSC to instances of $\mathsf{AMP}'(Q,\mathcal{A})$ in the same way as the one given in the reduction for Theorem 23. (b) Function g is defined as follows. For each instance (C,S) of MSC, let $f(C,S)$ be an instance $(Q,\mathcal{A})$ of oAMP$'$. Let $\mathcal{A}' \subseteq \mathcal{A}$ be a feasible solution to $\mathsf{oAMP}'(Q,\mathcal{A})$, *i.e.,* Q is covered by $\mathcal{A}'$. We define $g((C,S),\mathcal{A}')$ to be the following: for each $R_i(A \to (B_1,B_2,B_3),1)$ in $\mathcal{A}'$, we include C_i that contains elements encoded by B_1, B_2 and B_3. (c) Let $\alpha = \beta = 1$.

One can readily verify that f and g form a L-reduction with α and β from MSC to oAMP$'$. It is known that if there exists a L-reduction (f,g) from problem P_1 to problem P_2 with constants α_1 and α_2, and there exists PTIME ε-approximation algorithm for P_2 ($\varepsilon > 1$), then there exists PTIME $\alpha_1\alpha_2(\varepsilon-1)$-approximation algorithm for P_1 [Pap94]. Since there exists $c > 0$ such that there exists no $c \log |C|$-approximation algorithm for MSC (cf. [ACG+99]), $\mathsf{oAMP}'(Q,\mathcal{A})$ cannot be approximated within $c \log |X_Q \setminus X_C^Q|$ for constant $c > 0$ (observe that $|C| = |X_Q \setminus X_C^Q|$ in the reduction). □

(3) Finally, we consider the two special cases of the problem. We first show the NP-hardness. Observe that the NP-hardness proof in (1) for $\mathsf{dAMP}(Q,\mathcal{A},K)$ also works for the acyclic case, since the reduction constructs Q and $\mathcal{A}$ that form an acyclic $\mathcal{G}_{Q,\mathcal{A}}$ hypergraph.

Below we provide an NP lower bound proof for the elementary case, by reduction from the VERTEX COVER problem, denoted by VC. It is known that VC is NP-hard (cf. [Pap94]). Given an instance of VC, *i.e.,* a graph $G(V,E)$ and a natural number k, it is to find a vertex cover for G, *i.e.,* a subset $V' \subseteq V$ such that for each edge $(u,v) \in E$, at least one of u and v belongs to V' and $|V'| \leqslant k$.

Given an instance $G(V,E)$ and k of VC, we construct an instance of $\mathsf{dAMP}(Q,\mathcal{A},K)$, *i.e.,* a relational schema $\mathcal{R}$, an access schema $\mathcal{A}$ over $\mathcal{R}$, an RA query Q over $\mathcal{R}$ and a natural number k'. We show that there exists a subset $\mathcal{A}' \subseteq \mathcal{A}$ with $\sum_{R(X \to Y),N} N \leqslant k'$ if and only if there exists $\mathcal{A}_m \subseteq \mathcal{A}$ with $\sum_{R(X \to Y,N} N \leqslant k'$ and Q is covered by $\mathcal{A}_m$.

The reduction is given as follows. (a) The relational schema $\mathcal{R}$ consists of n relation schemas $R_i(A_1,A_2,B,C)$ for $i \in [1,n]$, where $n = |E|$, the number of edges in G. Intuitively, for each $i \in [1,n]$, (i) $R_i(A_1,A_2,B,C)$ encodes an edge e_i of E; and (ii) $R_i[A_1]$ and $R_i[A_2]$ encode the two vertices attached to e_i. (b) The access schema $\mathcal{A}$ consists of $5n$ constraints. For each $i \in [1,n]$, 5 constraints are defined: $R_i(\emptyset \to A_1, 5n), R_i(\emptyset \to A_2, 5n)$ $R_i(A_1 \to B, 1), R_i(A_2 \to B, 1), R_i((A_1,A_2,B,C) \to (A_1,A_2,B,C), 1)$. Intuitively, the first two constraints are to cover attributes that encode vertices, where $5n$ is to ensure we have to find $\mathcal{A}_m$ from $\mathcal{A}$ by removing such constraints. The third and fourth constraints indicate that each of the two vertices of an edge alone covers the edge. The fifth one ensures that the relation encoding the edge is indexed. As will be shown in Q, with $R_i[C]$, this one cannot be removed from $\mathcal{A}$ in order to keep Q covered. Observe that $\mathcal{A}$ is elementary. (c) We define an RA query Q (in fact in SPC) as follows:

$$\pi_Y \sigma_F (R_1(A_1,A_2,B,C) \times \cdots \times R_n(A_1,A_2,B,C)),$$

where (i) $Y = \{R_1[A_1], R_1[A_2], R_1[B], R_1[C], \ldots, R_n[A_1], R_n[A_2], R_n[B], R_n[C]\}$, and (ii) $F = F_1 \wedge F_2$, in which

- $F_1 = R_1[C] = 1 \wedge \cdots \wedge R_n[C] = 1$; and
- F_2 is a conjunction of $R_i[A_p] = R_j[A_q]$ $(i,j \in [1,n]$ and $p,q \in [1,3])$, where $R_i[A_p] = R_j[A_q]$ is in F_2 iff $R_i[A_p]$ and $R_j[A_q]$ encode the same vertex in V of G, *i.e.,* the edges encoded by R_i and R_j share the same vertex encoded by $R_i[A_p]$ $(R_j[A_q])$

Intuitively, (1) Y ensures that X_Q contains attributes in all relation schemas; (2) F_1 sets $X_C^Q = \{R_i[A_1], R_i[A_2], R_i[C] \mid i \in [1,n]\}$, enforcing that $R_i[B]$ has to be covered via X_C^Q and $\mathcal{A}$ if we want Q to be covered by $\mathcal{A}$; and (3) the indexing constraints for each R_i cannot be removed. Observe that Q is covered by $\mathcal{A}$. (d) Finally, we let $k' = 5kn + 3n$.

We next show that there exists a vertex cover $V' \subseteq V$ with $|V'| \leqslant k$ if and only if there exists $\mathcal{A}_m \subseteq \mathcal{A}$ with $\sum_{R(X \to Y,N} N \leqslant k'$ and Q is covered by $\mathcal{A}_m$.

$\boxed{\Rightarrow}$ First assume that there exists a vertex cover $V' \subseteq V$ with $|V'| \leqslant k$. Then define $\mathcal{A}_m \subseteq \mathcal{A}$ as follows: (i) for each v in V', suppose that $R_i[A_p]$ $(i \in [1,n], p \in [1,2])$ encodes v (while there may be multiple i and p such that $R_i[A_p]$ encodes v, we just keep one pair), $R_i(\emptyset \to A_p, 5n)$ is in $\mathcal{A}_m$; and (ii) for each R_i $(i \in [1,n])$, $R_i(A_1 \to B, 1)$, $R_i(A_2 \to B, 1)$ and $R_i((A_1,A_2,B) \to (A_1,A_2,B), 1)$ are in $\mathcal{A}_m$. One can verify that by the construction of F_2, Q is still covered by $\mathcal{A}_m$, and $\sum_{R(X \to Y,N) \in \mathcal{A}_m} N = 5kn + 3n \leqslant k'$.

$\boxed{\Leftarrow}$ Conversely, assume that there exists no vertex cover $V' \subseteq V$ with $|V'| \leqslant k$. That is, for any subset $V' \subseteq V$ with $|V'| \leqslant k$, there exists edge e_i in E such that V' cannot cover e_i. In other words, to ensure that $R_i[B] \in \mathsf{cov}(Q, \mathcal{A}_m)$ for each $i \in [1,n]$ with some

$\mathcal{A}_m \subseteq \mathcal{A}$, $\mathcal{A}_m$ has to include more than k access constraints of the form $R_i(\emptyset \to A_p, 5n)$ ($p \in [1,2]$, $i \in [1,n]$), as $\mathcal{A}_m$, together with Q, has to cover all $R_i[B]$ attributes in Q. Thus, $\sum_{R(X\to Y,N)\in\mathcal{A}_m} N > 5n * k$, *i.e.,* larger then k'. That is, there exists no $\mathcal{A}_m \subseteq \mathcal{A}$ with $\sum_{R(X\to Y,N)\in\mathcal{A}_m} N \leqslant k'$ such that Q is covered by $\mathcal{A}_m$.

We next show that oAMP$(Q,\mathcal{A})$ is not in APX. Observe that the L-reduction from MSC given in the proof of Theorem 23(2) is actually also a PTAS-reduction, which preserves the membership of APX class (cf. [ACG+99]). Moreover, the reduction constructs acyclic instances for oAMP. As MSC is not in APX, neither is oAMP in the acyclic case.

For the elementary case, we show it by a PTAS-reduction that revises the L-reduction for Theorem 23(2) as follows. (a) Function f maps an instance (C,S) to an instance of oAMP as follows. (i) Relational schema $\mathcal{R}$ consists of n relation schemas $R_i(A,B,B_1,B_2,B_3,E)$, *i.e.,* there exists a new attribute B to encode the subset C_i in C. (ii) Access schema $\mathcal{A}$ consists of $5n$ constraints. For each $i \in [1,n]$, 5 constraints are defined: $R_i(A \to B, 5n)$, $R_i(B \to B_1, 1)$, $R_i(B \to B_2, 1)$, $R_i(B \to B_3, 1)$, $R_i((A,B,B_1,B_2,B_3,E) \to (A,B,B_1,B_2,B_3,E), 1)$. (iii) Query Q remains the same as the one given in the proof for Theorem 23(2). (b) Function g maps feasible solutions $\mathcal{A}'$ to instances of oAMP to MSC as follows. For each $R_i(A \to B, 5n)$ in $\mathcal{A}'$, we include C_i in the cover C' for MSC. Similar to the acyclic case, one can readily verify that (f,g) is a PTAS-reduction from MSC to oAMP. Observe that for any instance (C,S) of MSC, instance $f(C,S)$ for oAMP is elementary. Since MSC is not in APX, neither is oAMP in the elementary case. □

3.4.2 Approximation Algorithms

Theorem 23 tells us that for AMP$(Q,\mathcal{A})$, any efficient algorithm is necessarily heuristic. Below we provide an efficient heuristic that *guarantees* to find a *minimal* $\mathcal{A}_m \subseteq \mathcal{A}$ that covers Q, *i.e.,* removing any constraint from $\mathcal{A}_m$ makes Q not covered by $\mathcal{A}_m$. Moreover, for the two special cases, there are approximation algorithms with approximation bounds.

Theorem 26: *(1) There is an algorithm for* AMP$(Q,\mathcal{A})$ *that finds minimal* $\mathcal{A}_m$ *in* $O(|Q|^2 + \|\mathcal{A}\|(|Q| + |\mathcal{A}|))$ *time.*

(2) For acyclic $(Q,\mathcal{A})$*,* oAMP$(Q,\mathcal{A})$ *is approximable within* $O(1 + |X_Q \setminus X_C^Q|)$ *in* $O(|Q| + |\mathcal{A}|)$ *time.*

(3) For elementary $Q(Q,\mathcal{A})$, $\mathsf{oAMP}(Q,\mathcal{A})$ *is approximable within* $O(1+|X_Q \setminus X_C^Q|^{\varepsilon})$ *in* $O((|Q|+|\mathcal{A}|)^{\frac{1}{\varepsilon}}|X_Q \setminus X_C^Q|^{\frac{2}{\varepsilon}}$ *time, for any constant* $\varepsilon > 0$. □

As a proof, we outline the algorithms as follows.

General case. As a proof of Theorem 26(1), we give an algorithm for $\mathsf{AMP}(Q,\mathcal{A})$ for the general case, denoted by minA (not shown). It is based on the following heuristics: a constraint $\phi = R(X \to Y, N_\phi)$, if it is not used to index a relation (Section 3.1), then it is less likely in the optimum solution $\mathcal{A}_m$ if Q remains covered by $\mathcal{A} \setminus \{\phi\}$, and moreover, (a) $\mathsf{cov}(Q,\mathcal{A}) \setminus \mathsf{cov}(Q,\mathcal{A} \setminus \{\phi\})$ is small; and (b) N_ϕ is large.

Based on this, algorithm minA works as follows. It first constructs the set $\Sigma_{Q,\mathcal{A}}$ of induced FDs of Q and $\mathcal{A}$. It then iteratively removes "redundant" FDs from $\Sigma_{Q,\mathcal{A}}$. In each iteration, it greedily selects an induced FD that corresponds to access constraint ϕ, such that (a) Q remains covered by $\mathcal{A} \setminus \{\phi\}$; and (b) $w(\phi) = \frac{c_1 \cdot N_\phi}{c_2 \cdot (|\mathsf{cov}(Q,\mathcal{A}) \setminus \mathsf{cov}(Q,\mathcal{A} \setminus \{\phi\})|+1)}$ is maximum among all constraints $\mathcal{A}$, where c_1 and c_2 are user-tunable coefficients for normalizing the numbers. It returns all access constraints corresponding to the remaining FDs in $\Sigma_{Q,\mathcal{A}}$ when it cannot remove more FDs from $\Sigma_{Q,\mathcal{A}}$.

Example 11: Consider Q_1 and $\mathcal{A}_0$ given in Example 5. Let $\mathcal{A}_1$ consist of all constraints in $\mathcal{A}_0$ and an additional ψ_5: $\mathsf{dine}((\mathsf{pid},\mathsf{year}) \to \mathsf{cid}, 366)$, which states that each person dines out at most 366 times per year. For $\mathsf{AMP}(Q_1,\mathcal{A}_1)$, algorithm minA finds that either ψ_2 and ψ_3, or ψ_3 and ψ_5 can be removed from $\mathcal{A}_1$ while keeping Q_1 covered. It then calculates $w(\psi_3) = \frac{c_1 \cdot 31}{c_2 \cdot 1}$ and $w(\psi_5) = \frac{c_1 \cdot 366}{c_2 \cdot 1}$. Suppose $c_1 = c_2 = 1$. Then minA greedily picks ψ_5 instead of ψ_2. It finds that no more constraints can be removed while keeping Q_1 covered. Thus minA returns $\mathcal{A}_m = \{\psi_1\ \psi_2, \psi_4\}$. □

Analysis. Algorithm minA always returns minimal $\mathcal{A}_m \subseteq \mathcal{A}$ for $\mathsf{AMP}(Q,\mathcal{A})$ since it keeps removing FDs until $\mathcal{A}_m$ is minimal. It can be implemented in $O(|Q|^2 + \|\mathcal{A}\|^2(|Q| + |\mathcal{A}|))$ time. Indeed, (1) it takes $O(|Q|^2 + |\mathcal{A}|)$ time to construct $\Sigma_{Q,\mathcal{A}}$; (2) it iterates at most $\|\mathcal{A}\|$ times; and (3) in each iteration, it takes $O(\|\mathcal{A}\| \cdot |\Sigma_{Q,\mathcal{A}}|) = O(\|\mathcal{A}\|(|Q| + |\mathcal{A}|))$ time to update the scores of all constraints in $\mathcal{A}$ and check whether removing each of them will make Q not covered. Therefore, algorithm minA takes $O(|Q|^2 + \|\mathcal{A}\|^2(|Q| + |\mathcal{A}|))$ time in total.

Acyclic case. To prove Theorem 26(2), we give an approximation algorithm, denoted by $\mathsf{minA}_{\mathsf{DAG}}$ (omitted), for the acyclic case of $\mathsf{AMP}(Q,\mathcal{A})$. It uses the following notion.

Weighted $\langle Q,\mathcal{A}\rangle$-hypergraph. For an RA query Q and an access schema $\mathcal{A}$, the *weighted $\langle Q,\mathcal{A}\rangle$-hypergraph* is a pair $(\mathcal{G}_{Q,\mathcal{A}}, w(\cdot))$, where $\mathcal{G}_{Q,\mathcal{A}} = (V,E)$ is the $\langle Q,\mathcal{A}\rangle$-hypergraph

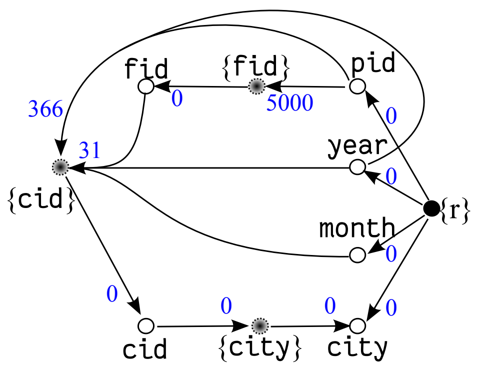


Figure 3.4: Weighted $\mathcal{G}_{Q_1,\mathcal{A}_1}$ for Q_1 and $\mathcal{A}_1$

for Q and $\mathcal{A}$, and $w(\cdot) : E \to \mathbb{N}^+$ assigns an natural number $w(e)$ to each hyperedge e in E. More specifically, $w(\cdot)$ is defined as follows. Recall the definition of $\mathcal{G}_{Q,\mathcal{A}}$ given in Section 3.3.2. For each induced FD $X \to Y$ in $\Sigma_{Q,\mathcal{A}}$ with $X = \{x_1, \ldots, x_p\}$ and $Y \setminus X = \{y_1, \ldots, y_q\}$, suppose that $X \to Y$ is derived from an constraint $R(X \to Y, N)$ in $\mathcal{A}$. Then

(i) $w(e_1) = N$, where e_1 is the hyperedge $(\{u_{x_1}, \ldots, u_{x_p}\}, u_Y)$ in $\mathcal{G}_{Q,\mathcal{A}}$ *w.r.t.* $X \to Y$;

(ii) $w(e_2) = \ldots = w(e_{q+1}) = 0$, for $e_2 = (\{u_Y\}, u_{y_1}), \ldots, e_{q+1} = (\{u_Y\}, u_{y_q})$; and

(iii) $w(\{r\}, u_A) = 0$ for all hyperedges emanating from the dummy node r of $\mathcal{G}_{Q,\mathcal{A}}$.

For instance, for Q_1 and $\mathcal{A}_1$ of Example 11, its weighted $\langle Q, \mathcal{A}\rangle$-hypergraph $\mathcal{G}_{Q_1,\mathcal{A}_1}$ is depicted in Fig. 3.4.

Algorithm minA$_{\mathsf{DAG}}$. Based on this notion, minA$_{\mathsf{DAG}}$ works as follows. It starts with node r in $\mathcal{G}_{Q,\mathcal{A}}$, and maintains a set S_G of nodes that are reachable from r via hyperpaths, along with the current shortest hyperpaths from r to them. It then iteratively explores the neighbors of S_G via breadth-first search (BFS), and updates S_G and the shortest hyperpaths from r to nodes in S_G accordingly. This is always feasible when $\mathcal{G}_{Q,\mathcal{A}}$ is acyclic. The iteration terminates when there is no more node to explore in $\mathcal{G}_{Q,\mathcal{A}}$. It then selects the subset Σ' of induced FDs in $\Sigma_{Q,\mathcal{A}}$ whose corresponding hyperedges are on the shortest hyperpaths from r to nodes denoting attributes in $(\hat{X}_Q \setminus \hat{X}_C^Q)$ (recall $\hat{X} = \rho_U(X)$ and Table 3.1). The algorithm returns access constraints corresponding to the induced FDs in Σ', plus one constraint with the minimum N to index each relation S in Q (Section 3.1).

Example 12: For Q_1 and $\mathcal{A}_1$ of Example 11, its weighted $\langle Q, \mathcal{A}\rangle$-hypergraph $\mathcal{G}_{Q_1,\mathcal{A}_1}$ in Fig. 3.4 is acyclic. Given Q_1 and $\mathcal{A}_1$, algorithm minA$_{\mathsf{DAG}}$ starts with r. It initializes S_G = $\{r\}$. It then iteratively searches neighbors of S_G via BFS. In the first round, it sets S_G = $\{u_{\mathsf{pid}}, u_{\mathsf{year}}, u_{\mathsf{month}}, u_{\mathsf{city}}\}$ and maintains their shortest hyperpaths to r as the edges. It keeps updating S_G by exploring the neighbors of S_G until no new edges can be explored.

There exist two hyperpaths from r to u_{cid}: one contains edge $(\{\mathsf{fid}, \mathsf{year}, \mathsf{month}\}, \mathsf{cid})$ with weight 31; and the other has edge $(\{\mathsf{pid}, \mathsf{year}\}, \mathsf{cid})$ of weight 5000; thus the former is the shortest path from r to u_{cid}. Algorithm $\mathsf{minA_{DAG}}$ returns access constraints ψ_1, ψ_2, ψ_4 in $\mathcal{A}_1$, which correspond to the edges in the shortest paths from r to u_{pid}, u_{fid}, u_{cid}, u_{month}, u_{year}, u_{city}. As Q_1 is already indexed by them, no more constraints are needed. □

<u>*Analysis*</u>. By Lemma 21, $\mathsf{minA_{DAG}}$ always returns $\mathcal{A}' \subseteq \mathcal{A}$ that covers Q. Let $c(\mathcal{A})$ denote the sum of the N's in all the constraints in $\mathcal{A}$. Let $\mathcal{A}'_I$ be the set of constraints in $\mathcal{A}'$ indexing a relation, and $\mathcal{A}'_{ni}$ be all the other constraints. Let $\mathcal{A}^{OPT}$ be the optimal solution to $\mathsf{AMS}(Q, \mathcal{A})$. We define $\mathcal{A}^{OPT}_I$ and $\mathcal{A}^{OPT}_{ni}$ analogous to $\mathcal{A}'_I$ and $\mathcal{A}'_{ni}$, respectively. Since $\mathsf{minA_{DAG}}$ selects constraints involved in shortest hyperpaths from r to $\rho_U(X_Q) \setminus \rho_U(X)$, we have the bound $\frac{c(\mathcal{A}')}{c(\mathcal{A}^{OPT})} = \frac{c(\mathcal{A}'_{ni})+c(\mathcal{A}'_I)}{c(\mathcal{A}^{OPT}_{ni})+c(\mathcal{A}^{OPT}_I)} \leqslant \frac{k*c(\mathcal{A}^{OPT}_{ni})+c(\mathcal{A}'_I)}{c(\mathcal{A}^{OPT}_{ni})+c(\mathcal{A}^{OPT}_I)} \leqslant k+1$, where $k = |\rho_U(X_Q) \setminus \rho_U(X)| \leqslant |X_Q \setminus X^Q_C|$.

Observe that the construction of the weighted hypergraph and the BFS search are both in $O(|\mathcal{G}_{Q,\mathcal{A}}|) = O(|Q| + |\mathcal{A}|)$ time. Therefore, algorithm $\mathsf{minA_{DAG}}$ is in $O(|Q| + |\mathcal{A}|)$ time.

Elementary case. As a proof of Theorem 26(3), we develop an algorithm, denoted by $\mathsf{minA_E}$ (omitted), for the elementary case $(Q, \mathcal{A})$ of $\mathsf{AMP}(Q, \mathcal{A})$. The idea is by reduction to the *directed minimum steiner arborescence* problem ($\mathsf{dminSAP}(G, u, V_T)$) (cf. [CCC+98]), which is to find the minimum weighed arborescence rooted at node u spanning all nodes in a set V_T in a weighted directed graph G.

The reduction is as follows. Given an elementary case $(Q, \mathcal{A})$, we construct an instance (G, u, V_T) of $\mathsf{dminSAP}$:

(a) G is the weighted $\langle Q, \mathcal{A}\rangle$-hypergraph $\mathcal{G}_{Q,\mathcal{A}_{ni}}$ for Q and $\mathcal{A}_{ni} \subset \mathcal{A}$, where $\mathcal{A}_{ni}$ consists of all those constraints in $\mathcal{A}$ that are not used to index a relation;

(b) u is the dummy node r in $\mathcal{G}_{Q,\mathcal{A}_{ni}}$; and

(c) V_T is the set of nodes in $\mathcal{G}_{Q,\mathcal{A}_{ni}}$ that correspond to attributes in $\hat{X}_Q \setminus \hat{X}^Q_C$ (see Table 3.1).

For elementary $(Q, \mathcal{A})$, $\mathcal{G}_{Q,\mathcal{A}_{ni}}$ is actually a weighted directed graph rooted at node r. Thus this is an instance of $\mathsf{dminSAP}(G, u)$. The reduction guarantees the following.

Lemma 27: *For elementary $(Q, \mathcal{A})$, $\mathsf{oAMP}(Q, \mathcal{A})$ has a c-approximation algorithm that takes $O(f(Q, \mathcal{A}))$-time if $\mathsf{dminSAP}(\mathcal{G}_{Q,\mathcal{A}_{ni}}, r, V)$ has a $(c-1)$-approximation algorithm in $O(f(|Q|, |\mathcal{A}_{ni}|) + |\mathcal{A}|)$-time.* □

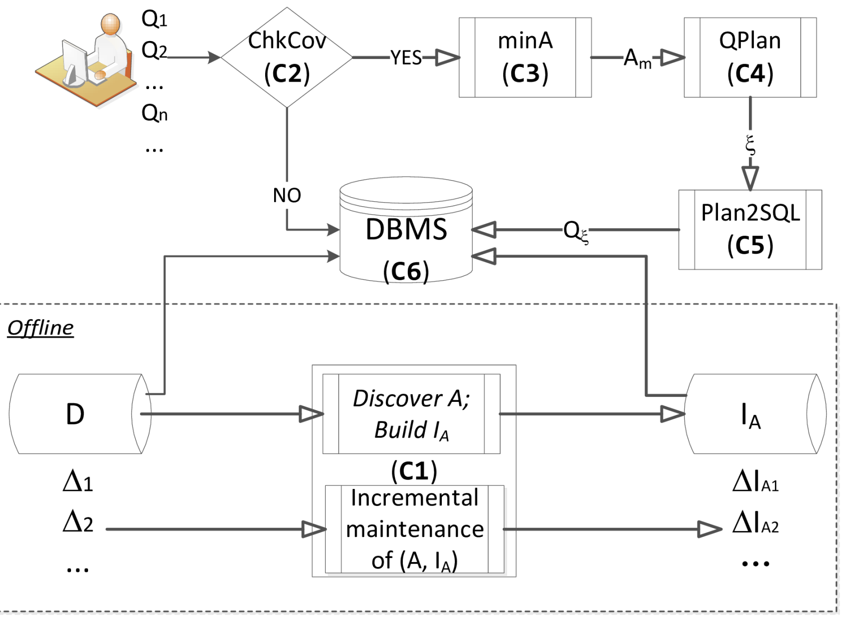


Figure 3.5: Bounded evaluability on DBMS

Proof: We prove Lemma 27 by showing step (c) of algorithm $\mathsf{minA_E}$ maps feasible solutions to $\mathsf{dminSAP}$ with approximation ratio c to feasible solutions to oAMP with approximation ratio $c+1$ in the elementary case. This can be verified along the same lines as the performance bound analysis of $\mathsf{minA_{DAG}}$ given in Section 3.4.2 and the proof of Lemma 25 above. □

Based on Lemma 27, algorithm $\mathsf{minA_E}$ works as follows. (a) It first builds the reduction described above. (b) It then computes the minimum weighted arborescence DT_r rooted at r of $\mathcal{G}_{Q,\mathcal{A}_{ni}}$ that spans all nodes in V_T, for $(\mathcal{G}_{Q,\mathcal{A}_{ni}}, r, V_T)$ constructed in (a). (c) It returns the following constraints in $\mathcal{A}$: (i) all constraints corresponding to edges in DT_r; and (ii) for each relation S in Q, one constraint for indexing S.

It is known that for $\mathsf{dminSAP}(\mathcal{G}_{Q,\mathcal{A}_{ni}}, r, V)$, there exists an $O(|V_T|^{\varepsilon})$-approximation algorithm that takes at most $O(|\mathcal{G}_{Q,\mathcal{A}_{ni}}|^{\frac{1}{\varepsilon}}|V_T|^{\frac{2}{\varepsilon}})$-time for any constant $\varepsilon > 0$ [CCC+98]. Moreover, observe that $|V_T| = |\hat{X}_Q \setminus \hat{X}_C^Q| \leqslant |X_Q \setminus X_C^Q|$. Thus $\mathsf{minA_E}$ is the algorithm promised by Theorem 26(3).

This completes the proof of Theorem 26.

3.5 Supporting Boundedly Evaluable Queries on DBMS

We next present a framework for incorporating bounded evaluation of RA queries into existing DBMS, based on covered queries. To simplify the discussion, we use $I_{\mathcal{A}}$ to denote the indices for all constraints in an access schema $\mathcal{A}$.

A framework of bounded evaluation. The framework is shown in Fig. 3.5. Given an application for queries over instances of a relational schema $\mathcal{R}$, it works as follows. As *offline* preprocessing (**C1** in Fig. 3.5), it discovers an access schema $\mathcal{A}$ from (sample) instances of $\mathcal{R}$, builds indices $I_{\mathcal{A}}$ for $\mathcal{A}$ on the instance D of $\mathcal{R}$ in use, and maintains $I_{\mathcal{A}}$ in response to updates to D. Given a user RA query posed on D, it first checks whether Q is covered by $\mathcal{A}$ (**C2**). If so, it picks a minimum set $\mathcal{A}_m$ of $\mathcal{A}$ that covers Q (**C3**), generates a bounded query plan ξ for Q under $\mathcal{A}_m$ (**C4**), and translates it into an SQL query Q_ξ (**C5**). Query Q_ξ can then be evaluated directly by the underlying DBMS on a bounded dataset D_Q identified by the bounded plan ξ (**C6**). If Q is not covered, it is executed against D by the DBMS. As will be seen shortly, a large fraction of RA queries are covered and hence, can be evaluated by accessing a small D_Q.

We next present its components in more details.

(1) Building and maintaining $\langle \mathcal{A}, I_{\mathcal{A}} \rangle$. It has three parts.

(a) Discovering $\mathcal{A}$. Like FDs, access constraints are defined on schema $\mathcal{R}$. They can be mined by extending dependency discovery tools [LLLC12], *e.g.,* TANE [HKPT99] for FDs. More specifically, on samples of a relation schema R, we search candidate attributes X and Y via revised FD mining, and use group_by on X and aggregates count on Y to form access constraint $R(X \to Y, N)$. These include those composed of attributes with a finite domain, *e.g.,* $R(X \to \mathsf{month}, 12)$, stating that a year has 12 months. These constraints hold on *all instances* of $\mathcal{R}$, just like discovered FDs.

Discovered constraints also include those determined by policies and statistics, *e.g.,* ψ_1 of Example 5 imposing a limit of 5000 friends per person, and one stating that US airports host carriers of at most 28 airlines (see Section 3.6). Such constraints *may* change if Facebook changes their policy or some US airports expand, and are thus maintained (see below).

(b) Building indices $I_{\mathcal{A}}$. For each discovered constraint $\phi = R(X \to Y, N)$ in $\mathcal{A}$, the index for ϕ is constructed by creating a table $T_{XY} = \pi_{XY}(D_R)$ and building a hash index on X over T_{XY}, where D_R is the instance of R in D. The index is no larger than $|D_R|$ and is constructed in $O(|D_R|)$ time. Thus, it takes $O(\|\mathcal{A}\||D|)$ time to build all indices in $\mathcal{A}$,

and the total size $I_{\mathcal{A}}$ is at most $O(\|\mathcal{A}\||D|)$.

(c) Incremental maintenance of $\langle\mathcal{A}, I_{\mathcal{A}}\rangle$. Now consider updates ΔD to D, *i.e.,* sequences of tuple insertions and deletions (which can simulate value modifications). We show that in response to ΔD, both constraints in $\mathcal{A}$ and indices $I_{\mathcal{A}}$ can be maintained by *bounded incremental* algorithms: their costs are determined by $\mathcal{A}$ and the size $|\Delta D|$ of updates ΔD only, and are independent of D and $I_{\mathcal{A}}$. In practice, ΔD is typically small, and hence so are the costs.

Proposition 28: *In response to updates ΔD to D, both $\mathcal{A}$ and $I_{\mathcal{A}}$ can be updated in $O(N_{\mathcal{A}}|\Delta D|)$ time, where $N_{\mathcal{A}} = \Sigma_{R(X \to Y,N) \in \mathcal{A}} N$.* □

(2) Checking whether Q is covered by $\mathcal{A}$. This can be carried out by algorithm CovChk of Section 3.2.

(3) Minimizing accessed data. This is conducted by the algorithms in Section 3.4 to minimize index access in $I_{\mathcal{A}}$.

(4) Generating boundedly evaluable query plans $\xi_{(Q,\mathcal{A})}$. This is done by using algorithm QPlan of Section 3.3.

(5) Interpreting $\xi_{(Q,\mathcal{A})}$ as SQL query Q_ξ. We develop an algorithm, denoted by Plan2SQL (omitted), to translate a bounded plan ξ into an SQL query Q_ξ, such that Q_ξ can be directly executed by DBMS. Given ξ and $\mathcal{A}$, Plan2SQL returns Q_ξ such that for any dataset $D \models \mathcal{A}$, Q_ξ returns $Q(D)$ by accessing the same amount of data *in index* $I_{\mathcal{A}}$ as ξ does in D. For instance, recall Q_1 and $\mathcal{A}_0$ of Example 5, $\mathcal{A}_0' = \mathcal{A}_0 \setminus \{\psi_3\}$, and the bounded query plan ξ for Q_1 under $\mathcal{A}_0'$ given in Example 5. Let the index relations in $I_{\mathcal{A}}$ under ψ_1, ψ_2 and ψ_4 in $\mathcal{A}_0'$ be ind_friend, ind_dine and ind_cafe, respectively. Plan2SQL$(\xi, \mathcal{A}_0')$ returns the following SQL query:

```
select distinct cid
from  ind_cafe
where city = NYC and cid in
          (select distinct cid
           from  ind_dine  /* no access to the underlying D */
           where month = MAY and year = 2015 and pid in
                     (select distinct fid from ind_friend where pid = p0))
```

(6) Query evaluation. This is simply done by issuing either Q_ξ or Q over $D_{\mathcal{A}}$ or D via DBMS, respectively.

Thus, bounded evaluation can be built on top of DBMS.

Added functionality. While indices and constraints are already employed by DBMS, their current mechanism stops short of taking advantage of bounded evaluation.

Indices and query plans. Query plans generated by conventional query engines fetch entire tuples first and then filter tuples based on the query (see, *e.g.,* [AHV95]), by employing *tuple*-based indices, *e.g.,* hash index and tree-based index [RG00]. In contrast, a boundedly evaluable query plan makes use of *attribute*-based indices. It identifies what attributes are necessarily needed, fetches values of the attributes, infers their connection with other attributes, composes attribute values into tuples and validates the tuples (via the indexing condition of Section 3.1). However, existing DBMS stops short of exploring this, no matter what indices are provided.

This observation is verified by examining system logs of commercial DBMS, which shows excessive duplicated and unnecessary attributes in tuples fetched by DBMS, and the redundancies get inflated rapidly when joins are involved.

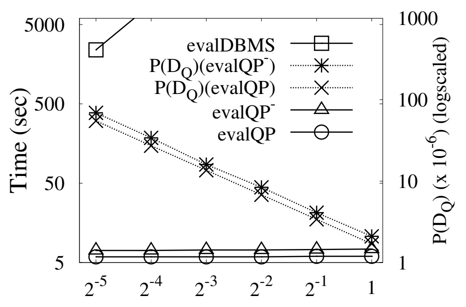
We also check whether a query Q is boundedly evaluable before Q is executed, as opposed to conventional DBMS.

(2) Constraints. Query optimization has been studied for reformulating a query Q as another query by "chasing" Q with constraints [AHV95, Pop01, ICDK14]. However, to the best of our knowledge, conventional query engines have made little use of it, partly because the chasing process is costly and may not even terminate. Moreover, cardinality constraints have not been explored for this purpose. In contrast, we use cardinality constraints to generate boundedly evaluable query plans, instead of query reformulation. These constraints are easy to reason about and can be readily supported by DBMS.

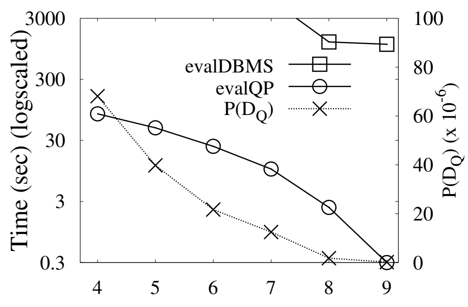
(3) Join ordering. Query engines may reorder joins in a query plan to minimize estimated data access [MFE13, GN14]. It is an effective optimization strategy complementary to this work. However, to comply with bounded data access via access constraints, some joins in a boundedly evaluable query plan cannot be reordered. It is an interesting topic to study what joins can be reordered in boundedly evaluable plans.

3.6 Experimental Study

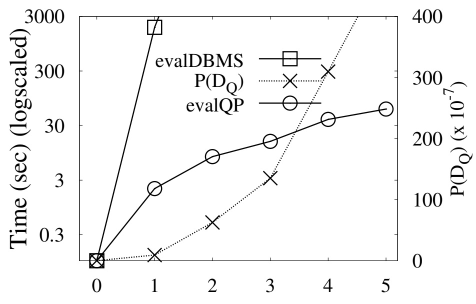
Using real-life data, we conducted two sets of experiments to evaluate (1) the effectiveness of the RA-query evaluation approach based on the bounded evaluability analysis,



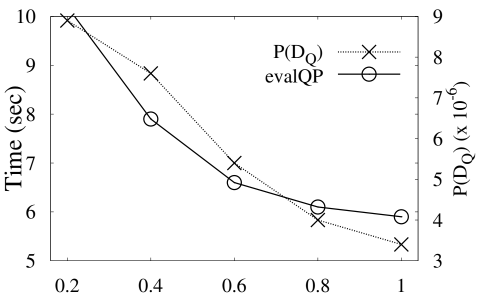
(a) AIRCA: varying $|D|(\times 60\text{GB})$



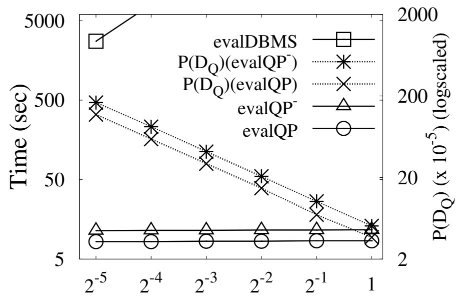
(b) AIRCA: varying #-sel



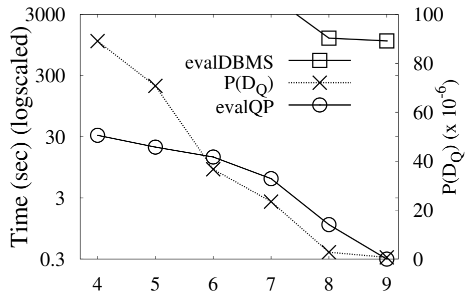
(c) AIRCA: varying #-join



(d) AIRCA: varying $\|\mathcal{A}\|(\times\|A\|)$



(e) TFACC: varying $|D|(\times 21.4\text{GB})$



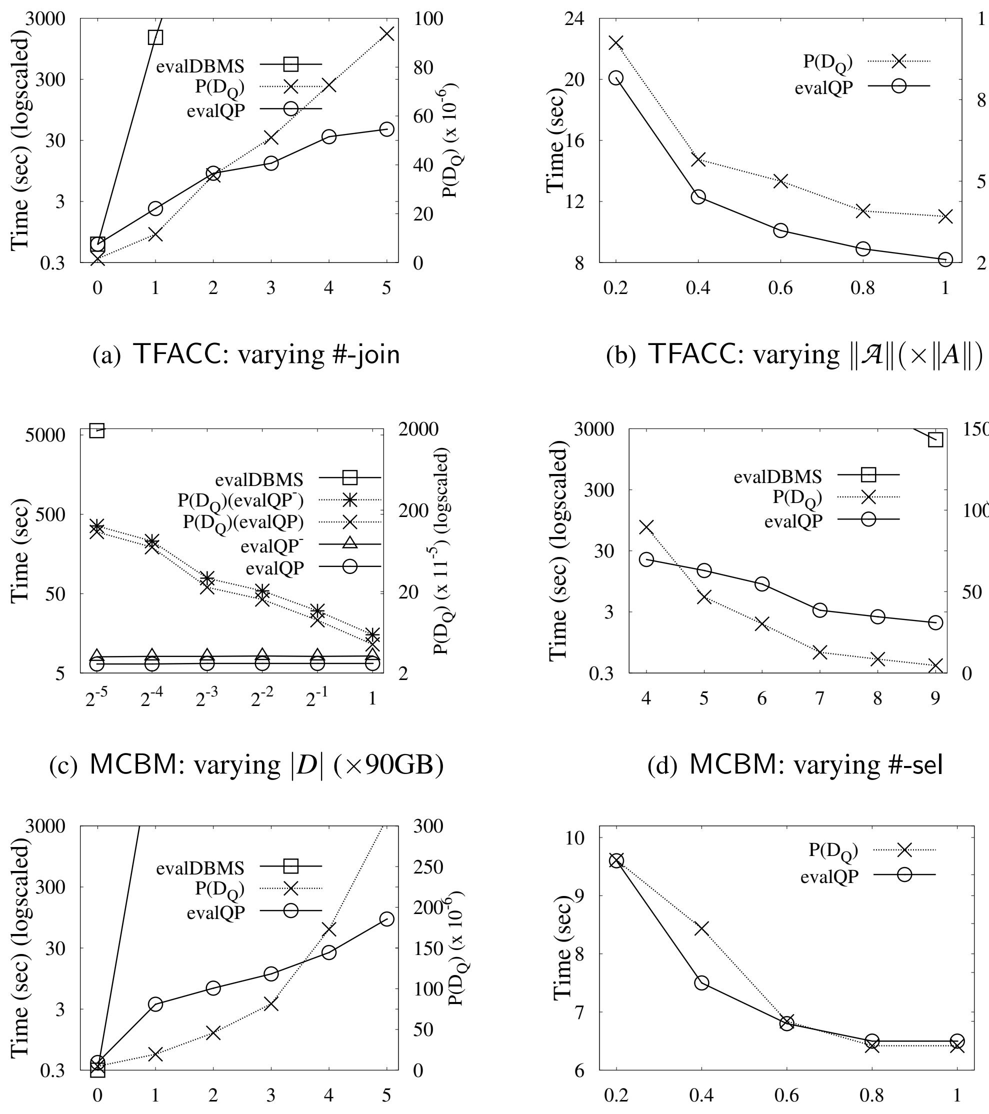
(f) TFACC: varying #-sel

Figure 3.6: Effectiveness of bounded evaluability

and (2) the efficiency of algorithms ChkCov, QPlan and minA.

Experimental setting. We used three datasets: two real-life (AIRCA and TFACC) and one benchmark (MCBM).

(1) US Air carriers (AIRCA) records flight and statistic data of certified US air carriers from year 1987 to 2014. It consists of Flight On-Time Performance data [BTSa] for departure and arrival data, and Carrier Statistic data [BTSb] for airline market and segment data of the air carriers. It has 7 tables, 358 attributes, and over 162 million tuples, about 60GB of data.



(a) TFACC: varying #-join

(b) TFACC: varying $\|\mathcal{A}\|(\times\|A\|)$

(c) MCBM: varying $|D|$ ($\times$90GB)

(d) MCBM: varying #-sel

(e) MCBM: varying #-join

(f) MCBM: varying $\|\mathcal{A}\|(\times\|A\|)$

Figure 3.7: Effectiveness of bounded evaluability (Cont.)

(2) UK traffic accident (TFACC) integrates the Road Safety Data [Gova] of road accidents that happened in the UK from 1979 to 2005, and National Public Transport Access Nodes (NaPTAN) [Govb]. It has 19 tables with 113 attributes, and over 89.7 million tuples in total, about 21.4GB of data.

(3) Mobile communication benchmark (MCBM) was generated by using a commercial benchmark from Huawei Technologies Co. Ltd. The dataset consists of 12 relations with 285 attributes, simulating mobile communication scenarios. In the tests, we varied the number of tuples from $2^{-5} \times 360$ to 360 million, and used 360 million by default,

about 90GB of data.

All of the three datasets were stored in MySQL.

Access schema. We extracted 266, 84 and 366 access constraints for AIRCA, TFACC and MCBM, respectively, by using the discovery method in Section 3.5. For example, a constraint on AIRCA is OnTimePerformance(Origin $\rightarrow$ AirlineID, 28), *i.e.,* each airport hosted carriers of at most 28 airlines. On TFACC, we had Accident((data, police_force) $\rightarrow$ accident_ID, 304), *i.e.,* each police force in the UK had handled no more than 304 accidents within a single day from 1979 to 2005. In fact there are many more access constraints in the datasets, which were not used in our tests. We built indices for the constraints by using DBMS (see Section 3.5).

RA *queries generator*. We generated queries by using attributes that occurred in the access constraints and constants randomly extracted for those attributes. For MCBM, the query generation also complied with the provided query templates. We generated 300 RA queries Q on these datasets, 100 for each. The queries vary in the number #-sel of equality atoms in the selection conditions in the range of [4, 9], #-join of joins in the range of [0,5] and #-unidiff of set difference and union operators in the range of [0, 5].

Algorithms. We implemented the following algorithms in Python: (1) ChkCov (Section 3.2) to check whether an RA query is covered; (2) QPlan (Section 3.3) to generate canonical query plans for covered queries; (3) minA, $\mathsf{minA}_{\mathsf{DAG}}$ and $\mathsf{minA}_{\mathsf{E}}$ (Section 3.4) to find minimum access constraints for covered queries; (4) Plan2SQL to interpret canonical query plans generated by QPlan as SQL queries (Section 3.5); (5) evalQP^{-} and evalQP to evaluate the translated queries Q_ξ with and without minimized $\mathcal{A}_m$ (via minA; by Plan2SQL) using DBMS, respectively; and (6) evalDBMS that directly uses DBMS engine for query evaluation, with a configuration in favor of DBMS, which is described as follows.

Configuration. For DBMS, we used MySQL 5.5.44 (MyISAM engine) and PostgreSQL 9.3.9. Both original queries and query plans generated by our algorithms are executed on the same database server. In favor of MySQL and PostgreSQL, we optimize both MySQL and PostgreSQL with extra indices in addition to access constraints for selectivity and joins. More specifically, we build the following additional indices for MySQL and PostgreSQL:

- for each access constraint $R(X \rightarrow Y, N)$ discovered on relation R, we build hash indices on attributes X, attributes Y and attributes XY;
- for each attribute A in the selection conditions of queries generated, we build a

% of constraints used in $\mathcal{A}$	25%	50%	75%	100%
#-covered(bounded) (AIRCA)	48(55)	56(59)	61(67)	61(70)
#-covered(bounded) (TFACC)	44(51)	50(57)	52(65)	52(65)
#-covered(bounded) (MCBM)	34(39)	40(42)	42(48)	42(48)

Table 3.2: # of covered (bounded) queries *w.r.t.* $|\mathcal{A}|$

hash index on A to speedup the selectivity operation for query plans generated by MySQL and PostgreSQL; and

◦ primary and foreign key indices and B-tree index on numerical attributes.

These were *all disabled* when testing our query plans.

The experiments were conducted on an Amazon EC2 d2.xlarge instance with 14 EC2 compute units and 30.5GB memory. All the experiments were run 3 times. The average is reported here.

Experimental Results. We next report findings. As results for PostgreSQL are even worse than MySQL when compared with ours, we mainly report MySQL to save space.

Exp-1: Effectiveness of bounded evaluability.

(I) Percentage of bounded evaluable and covered RA *queries*. Varying the number of access constraints, we tested the number of covered queries (via ChkCov) and boundedly evaluable queries (by manual examination). The results are shown in Table 3.2, and tell us the following. (a) When all the discovered constraints are used, (i) at least 70, 65 and 48 out of 100 queries are boundedly evaluable, and (ii) 61, 52 and 42 are covered, on AIRCA, TFACC and MCBM, respectively. That is, at least 70%, 65% and 51% of the queries are boundedly evaluable, and among them 87%, 80% and 87.5% are covered. Hence, covered queries are indeed effective for determining the bounded evaluability of RA queries. (b) The more access constraints are used, the more queries are covered and boundedly evaluable, as expected. Nonetheless, among all the covered queries, 78.7%, 84.6% and 80.9% are already covered by only 25% of the discovered access constraints on AIRCA, TFACC and MCBM, respectively. That is, a large number of queries can be covered by a small number of constraints.

(II) Effectiveness of covered queries. We next evaluated the effectiveness of query plans generated by QPlan, by comparing the run time of evalQP and evalDBMS, both executed by MySQL. The results are reported in Figures 3.6 and 3.7, on datasets AIRCA, TFACC and MCBM, by varying $|D|$, Q and $\|\mathcal{A}\|$. We report (i) the average evaluation

time (the left y-axis), and (ii) ratio $P(D_Q) = |D_Q|/|D|$, measuring the total amount of data D_Q accessed by our query plans (the right y-axis), which is assessed by using MySQL's EXPLAIN statement. Unless stated otherwise, we used the full-size datasets, all access constraints, and 5 covered queries randomly chosen.

(1) Impact of $|D|$. To evaluate the impact of $|D|$, we varied the datasets by using scale factors from 2^{-5} to 1. As shown in Figures 3.6(a), 3.6(e) and 3.7(c), the results tell us the following.

(a) The evaluation time of evalQP is indifferent to the size of D, as expected for covered queries. This verifies the property of covered queries.

(b) Bounded query plans work well with large D. Indeed, evalQP took less than 5.9s, 8.3s, 6.5s with MySQL, and 5.5s, 9.0s, 7.0s with PostgreSQL, on AIRCA, TFACC and MCBM, respectively, no matter how large the datasets were. In contrast, even on the smallest subsets with scale factor 2^{-5}, evalDBMS took 2398s, 2759s, 5675s by MySQL, and 3598s, 3851s, 7301s by PostgreSQL; it could not terminate within 2 hours for all larger subsets. This is why few points are reported for evalDBMS in the figures. In fact, evalDBMS could not finish within 14 hours on all three full-size datasets (both MySQL and PostgreSQL). That is, evalDBMS is at least 8.5×10^3, 6.1×10^3 and 7.8×10^3 times slower on AIRCA, TFACC and MCBM, respectively. The larger the dataset is, the bigger the gap between evalDBMS and evalQP is.

(c) Query plans generated by QPlan accessed a very small fraction of the data: $P(D_Q)$ is 1.7×10^{-6}, 3.7×10^{-5}, 2.2×10^{-6} on full-size AIRCA, TFACC and MCBM. *i.e.,* 0.00017%, 0.0037% and 0.00022% of these datasets, respectively.

Remark. As shown above, evalQP outperforms evalDBMS by at least 3 orders of magnitude, for reasons explained in Section 3.5. We also find that when queries Q use key attributes only, evalDBMS is as fast as evalQP if extra key/foreign key indices are built for MySQL and PostgreSQL, *e.g.,* less than 3s with one join on full AIRCA. However, as long as Q involves non-key attributes, evalDBMS performs poorly on big tables, even provided with all indices. It gets worse when the number of non-key attributes increases. By looking into MySQL's log and its EXPLAIN output, we verified that this is partially due to the following. Given an access constraint $R(X \rightarrow Y, N)$, evalQP fetches only distinct values of the relevant XY attributes, but evalDBMS fetches entire tuples with irrelevant attributes of R, although those attributes are not needed for answering Q at all, no matter what indices are provided. This led to duplicated (X,Y) values when X is not a key, and the duplication got rapidly inflated by joins, *e.g.,* EXPLAIN output shows that MySQL consistently accesses entire tables when there are non-key attributes.

(2) Impact of Q. To evaluate the impact of queries, we varied #-sel of Q from 4 to 9, #-join from 0 to 5 and #-unidiff of set operators (union and set-difference) from 0 to 5, while keeping the other factors unchanged.

The results are reported in Figures 3.6(b), 3.6(f), 3.7(d) for varying #-sel and Figures 3.6(c), 3.7(a), 3.7(e) for varying #-join. We find the following. (a) The complexity of Q has impacts on the query plans generated by QPlan. The larger #-sel or the smaller #-join is, the faster the query plans are, and the smaller data D_Q is accessed. This is because with more selections or fewer joins, our plans generated by QPlan took less steps to fetch all attribute values needed. (b) Algorithm evalQP scales well with #-sel and #-join. It found answers for largest Q within 89.5s, on the three full-size datasets. (c) Algorithm evalDBMS is almost indifferent to #-sel; in fact it only terminated within 3000s on extremely restricted (constant) selection queries, with at most one join on non-key attribute. But it is very sensitive to #-join: it did the best when #-join = 0, *i.e.,* if there is no join (or Cartesian product) at all; but it cannot finish the job within 3000s for queries with 2 joins on all three datasets.

Our query plans are indifferent to #-unidiff (hence the results are not shown). This is because our query plans fetch data via max SPC sub-queries, independent of the number of union and set-difference operations in the queries. We do not report the results of evalDBMS since it did not complete its computation within 3000s on all three datasets.

(3) Impact of $\|\mathcal{A}\|$. To evaluate the impact of access constraints, we varied $\|\mathcal{A}\|$ with scale factors from 0.2 to 1 in 0.2 increments, and tested the queries that are covered. Accordingly we varied the indices used by evalDBMS. We report $P(D_Q)$ and run time of evalQP. As shown in Figures 3.6(d), 3.7(b) and 3.7(f), (a) more constraints help QPlan generate better query plans, as expected. For example, when all access constraints were used, evalQP took 5.8s, 8.5s and 6.3s for queries on AIRCA, TFACC and MCBM, respectively, while they took 10.2s, 20.1s and 9.6s given 20% of the constraints. (b) The more access constraints are used, the smaller $|D_Q|$ is, as QPlan can find better plans given more options. (c) Algorithm evalDBMS did not produce results in any test within 3000s, even given the indices in full-size $\mathcal{A}$ of constraints.

(III) Effectiveness of minA. We also evaluated the effectiveness of minA for minimizing access schemas by comparing evalQP and evalQP$^-$. As reported in Figures 3.6(a),

3.6(e) and 3.7(c), (1) minA helps QPlan generate query plans that access less data; indeed, evalQP accessed much smaller D_Q than evalQP$^-$ in most cases; for example, $P(D_Q)$ is 0.0037% for evalQP on full-size TFACC, while it is 0.0051% for evalQP$^-$; and (2) minA also enables query plans to use indices of smaller size (*i.e.,* index relations; not shown). For example, on full-size TFACC, evalQP used index no larger than 2.1% of the size of D while it was 3.3% for evalQP$^-$.

(IV) Size and creation time of indices. The total indices for all access constraints are of 7.7GB, 3.6GB and 9.5GB, accounting for 12.8%, 16.8% and 10.6% of $|D|$. They are smaller than the bound estimated in Section 3.5, since many constraints use attributes with small domains. They took 3.1, 2.2 and 4.2 hours to build offline for AIRCA, TFACC and MCBM, respectively, and were used to answer all queries.

Exp-2: Efficiency. The second set of experiments evaluated the efficiency of our algorithms ChkCov, QPlan, minA, minA$_{\mathsf{DAG}}$ and minA$_{\mathsf{E}}$ on queries and access schemas for each of AIRCA, TFACC and MCBM. We found that ChkCov, QPlan, minA, minA$_{\mathsf{DAG}}$ and minA$_{\mathsf{E}}$ took at most 65ms, 99ms, 86ms, 84 ms and 74 ms, respectively, for all queries on three datasets, with all the access constraints.

Summary. From the experiments we find the following. (1) Covered queries give us a practical effective syntax for boundedly evaluable RA queries. Over 80% of boundedly evaluable queries are covered. (2) Bounded evaluability is promising for querying large datasets. Indeed, (a) it is easy to find access constraints from real-life data, and many queries are covered under a small number of such constraints; and (b) for covered queries, the evaluation time and the amount of data accessed are *independent of* the size of the underlying dataset. As a result, on a real-life dataset of 60GB, evalQP answers queries in 5.9 seconds by accessing at most 0.00017% of the data on average, while evalDBMS is unable to find answers within 3000 seconds even on a dataset of 3.75 GB, with even more indices than evalQP can use. The performance gap between evalQP and evalDBMS gets bigger on larger datasets. (3) The size of the indices needed is 13.4% of $|D|$ on average. (4) Our algorithms are efficient: they take at most 0.2 second in all cases tested.

Summary

We have proposed a feasible solution to make effective use of bounded evaluability. Our solution consists of both fundamental results (the existence of an effective syntax and the minimality of access constraints), and efficient algorithms (for checking query coverage, generating bounded query plans, and minimizing access schema). Our experimental results have shown that it is promising to make practical use of bounded evaluability. Indeed, a large number of RA queries are covered, and covered queries can be efficiently evaluated without worrying about the size of the underlying datasets.

Discussion. With the detailed development of bounded evaluability given above, we finally discuss the reasons behind the performance of bounded plans, the limitation of bounded evaluability with possible solutions, and the relationships between boundedly evaluable plans and existing techniques on query evaluation and optimization.

The elevator pitch behind the performance speedup of boundedly evaluable query plans against conventional ones is straightforward: boundedly evaluable query plans achieve guaranteed data-independent scale independence, by fetching and maniputating *deduplicated* values instead of full tuples. More specifically, the bounded evaluation framework is able to identify the cases that boundedly evaluable query plans can apply, by inference with queries and access schema. The inference gives us bounded plans, which run over projected relations $R[XY]$ of relations R for access constraints $R(X \to Y, N)$, on which hash indices are built with X as the keys.

The effectiveness of boundedly evaluable query plans highly depends on the availability of access constraints, which varies case by case. However, bounded evaluability can be better used for applications with fixed query workload while the datasets are updated frequently, *e.g.,* searches over e-commercial platforms. Indeed, given a query workload, we can examine the datasets and discover a set of access constraints relevant to the query workload, possibly with larger N's as long as they are not heavily used in the bounded plans. This will also reduce the space cost for storing the indices of access constraints as only relevant ones need to be maintained. In addition, in many cases query plans that are not boundedly evaluable may contain sub-plans that are bounded. These sub-plans can be answered scale independently, to reduce the evaluation time of conventional plans over big datasets.

The idea of bounded query plans (*i.e.,* answering queries using access constraints) is similar to query evaluation over column-stores [AMH08] and index-only scans [Pos]. The major difference between bounded evaluation and column stores is that, column stores map attribute values of columns to tuple IDs in the relation (cf. [AMH08]), while indices for access constraints maps *distinct* attribute values (*e.g.,* on attributes X) to some other *distinct* attribute values (*e.g.,* on attributes Y). The latter enables deduplication over attribute values while the former cannot, due to the need to store tuple IDs. The difference makes bounded plan scale independent while column stores are not.

Closer to bounded evaluability is the idea of index-only scans [Pos]. Indeed, bounded plans are a special case of index-only query plans that manipulate distinct data values to achieve scale independent, which are not guaranteed by generic query plans based on index-only scans.

Part II

Beyond Boundedly Evaluable Queries

In Part I, we have studied bounded evaluability from both theoretical and practical aspects. While they are effective in querying big data, in practice, there are still many queries that are not boundedly evaluable. In this part, we study such queries by extending the idea of bounded evaluability. We propose two methods: resource-bounded approximation in Chapter 4 and bounded query rewriting using views in Chapter 5.

Resource-bounded approximation extends bounded evaluability such that under an access schema $\mathcal{A}$, given big data D and query Q, while we cannot compute exact answers $Q(D)$ within resource that is independent of the scale of D, there exists access schema $\mathcal{A}'$ that extends $\mathcal{A}$ which holds on D, such that for any resource ratio $\alpha \in (0,1]$, we can compute a set S of approximate answers to Q in D and a deterministic accuracy bound η, by accessing no more than $\alpha|D|$ tuples in D via $\mathcal{A}'$, such that S has accuracy at least η.

Bounded query rewriting using views combines bounded evaluability with materialized views. A query Q has a *bounded rewriting* using a set of views if there exists a query Q' expressed in the same language as Q, such that given a dataset D, $Q(D)$ can be computed by Q' that accesses only cached views and a small fraction D_Q of D. Again, we consider datasets D that satisfy an access schema $\mathcal{A}$, such that the size of D_Q and the time to identify D_Q are independent of the size of D, no matter how big D is.

Resource-bounded approximation and bounded query rewriting using views provide an effective approach to evaluating those queries that are not boundedly evaluable, by making use of the idea of bounded evaluability.

Chapter 4

From Bounded Evaluability to Bounded Approximation

In this chapter, we propose a resource-bounded framework for answering queries in relational algebra. It is parameterized with a resource ratio $\alpha \in (0,1]$ indicating that we can only afford to access an α-fraction of a big dataset D, given bounded resources such as time constraint and available processors. For all queries Q posed on D, it returns exact answers $Q(D)$ if they can be computed by accessing a bounded amount of data. Otherwise, it computes approximate answers with a deterministic accuracy bound η, by accessing an α-fraction of D. Underlying the framework are (1) an access schema, which helps us identify data needed to answer Q and fetch the data from D, (2) an accuracy measure to assess approximate answers, in terms of their relevance to users' need and their coverage of the exact answers, and (3) algorithms for computing exact and approximate answers with bounded resources. The framework can be extended to answer relational queries with aggregate functions. Using real-life and synthetic data, we experimentally verify the effectiveness, scalability and accuracy of the framework.

In Chapter 3, we have shown using several real-life datasets that under a few hundreds access constraints, 67% of RA queries of a workload we used are bounded; their query plans outperform commercial DBMS by 3 orders of magnitude, and the gap gets larger on bigger D.

But what can we do about the 30+% of RA queries that are not boundedly evaluable? Can we answer such queries with bounded resources? We study this issue in this chapter, by extending the bounded evaluation framework developed in Chapter 3.

Resource-bounded approximation. We propose an approximation scheme to answer queries that are not bounded. It is parameterized with a *resource ratio* $\alpha \in (0,1]$, indicating that our available resources allow us to only access an α-fraction of a big dataset D. Underlying the scheme are (a) an access schema $\mathcal{A}$ that combines cardinality constraints and indices to representative tuples in D, and (b) an accuracy measure to assess approximation answers.

Given any RA query Q and instance D of $\mathcal{R}$ that satisfies $\mathcal{A}$, the scheme is to find a set S of tuples (approximate query answers) and a provable *accuracy bound* η such that

- it accesses a fraction D_Q of D with $|D_Q| \leqslant \alpha|D|$, and
- the accuracy bound of S is at least η.

That is, we compute S by accessing an α-fraction D_Q of D, identified via guided search by access schema. The bound η is *deterministic*: *each* tuple $s \in S$ is a sensible answer to Q in D, and *for each* $t \in Q(D)$, there exists $s \in S$ that is close enough to t, above η. That is, S is "relevant" to what users want to find, and it "covers" exact answers $Q(D)$.

We show an *approximability result*: for any database schema $\mathcal{R}$, we can extend access constraints to an access schema $\mathcal{A}$ such that for any instance D of $\mathcal{R}$, (a) D satisfies $\mathcal{A}$, and (b) for any resource ratio $\alpha \in (0,1]$ and RA query Q, a set of approximate answers can be computed with a bound η, by accessing D_Q via $\mathcal{A}$, where $|D_Q| \leqslant \alpha|D|$. In our experiments we find that $\eta \geqslant 0.82$ when α is *as small as* 5.5×10^{-4}.

Resource-bounded framework. Putting the approximation scheme and bounded evaluation in Chapter 3 together, we propose a framework for answering RA queries with bounded resources. Given an access schema $\mathcal{A}$, a resource ratio α, an instance D of $\mathcal{R}$ that satisfies $\mathcal{A}$, and an RA query Q,

(1) it checks whether Q is boundedly evaluable under $\mathcal{A}$;

(2) if so, it computes $Q(D)$ by accessing bounded $D_Q \subseteq D$;

(3) otherwise, it identifies D_Q with $|D_Q| \leqslant \alpha|D|$ by using $\mathcal{A}$, and computes $Q(D_Q)$

with a deterministic accuracy bound η based on our approximation scheme.

Methods for steps (1) and (2) have been developed in Part I. We focus on step (3) in this chapter, the approximation scheme.

Example 13: Consider a database schema $\mathcal{R}_0$ with three relations: (a) $\mathsf{person}(\mathsf{pid}, \mathsf{city})$, stating that pid lives in city, (b) $\mathsf{friend}(\mathsf{pid}, \mathsf{fid})$, saying that fid is a friend of pid, and (c) $\mathsf{poi}(\mathsf{id}, \mathsf{type}, \mathsf{city}, \mathsf{street})$, stating that a POI id is of type and is on street of city. Access constraints over $\mathcal{R}_0$ include

- ψ_1: $\mathsf{friend}(\mathsf{pid} \to \mathsf{fid}, 5000)$,
- ψ_2: $\mathsf{person}(\mathsf{pid} \to \mathsf{city}, 1)$.

Here ψ_1 is a constraint imposed by Facebook [GBDS14]: a limit of 5000 friends per person; and ψ_2 states that each person lives in at most one city. An index is built for ψ_1 such that given a pid, it returns all fid*s* of pid from friend; similarly for ψ_2. We denote by $\mathcal{A}_c^1$ the set consisting of ψ_1 and ψ_2.

(1) Consider query Q_1 to *find the cities where my friends live*, from Graph Search of Facebook [Faca], written in SQL:

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

where p_0 indicates "me". When an instance D_0 of $\mathcal{R}_0$ is "big", *e.g.,* Facebook has billions of users and trillions of friend links [GBDS14], it is costly to compute $Q_1(D_0)$. However, we can do better since Q_1 is boundedly evaluable under $\mathcal{A}_c^1$: (a) we identify and fetch at most 5000 fid*s* for p_0 from friend by using the index for ψ_1, and (b) for each fid fetched, we get his city by fetching 1 tuple from person via the index for ψ_2. In total we fetch a set D_Q of 10,000 tuples, instead of trillions; it suffices to compute $Q_1(D_0)$ by using D_Q (cf. Chapter 3).

(2) Consider query Q_2 to *find me the streets in a city where a friend of mine lives and on which there is a cinema*:

```
select  c.street
from    friend as f,  poi as c,  person as p
where   f.pid = p0 and f.fid = p.pid and
        p.city = c.city and c.type = "cinema"
```

In contrast to Q_1, Q_2 is not boundedly evaluable under $\mathcal{A}_c^1$. Nonetheless, as will be shown later, given a resource ratio α, our resource-bounded approximation scheme

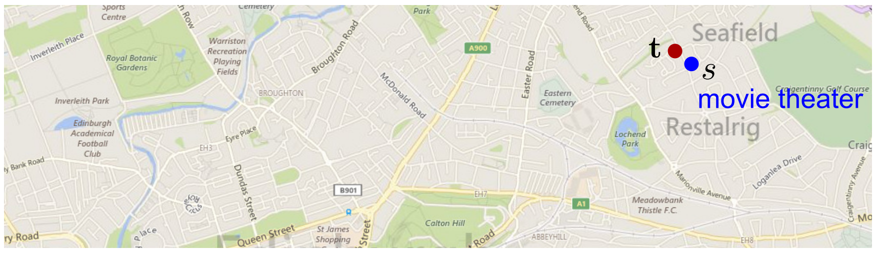


Figure 4.1: Approximate answer vs. exact answer

is able to find a set S of approximate answers by accessing at most $\alpha|D_0|$ tuples. It guarantees that S is accurate, *e.g.,* $s \in S$ is a "movie theater" and is hence in p_0's interest; moreover, each cinema $t \in Q_2(D)$ is either in S or close to some $s' \in S$. moreover, it is within 0.1 miles of a cinema (exact answer) $t \in Q_2(D)$. Figure 4.1 shows an approximate answer s when α is 10^{-4}. It is a "movie theater" and is hence in p_0's interest; moreover, every exact answer in $Q_2(D_0)$ is close to an answer in S, *e.g.,* cinema t is within 0.1 miles of s in Fig. 4.1.

The approximate answers give us an accurate and quick estimate of the exact answers. They suffice for exploratory queries for, *e.g.,* real-time problem diagnosis on logs [AMP+13]. Moreover, when we do not have considerable background knowledge about the semantics of a dataset, query formulation is a "trial and fail" process; approximate answers help us identify what we want to find, and suggest how to our queries. Better yet, they are often accurate enough for us to accept as sensible answers in real life. □

Overview. We propose a framework for answering relational queries with bounded resources, and develop foundation and algorithms underlying the framework.

(1) We extend access schema in Chapter 2 to help us compute exact and approximate answers uniformly (Section 4.1).

(2) We propose an accuracy measure to assess the accuracy of approximate answers (Section 4.2). As opposed to prior metrics, it evaluates answers in two aspects: (a) the *relevance*, *i.e.,* how close each tuple $s \in S$ is to user's interest; and (b) the *coverage*, *i.e.,* whether S covers the corresponding exact answers, via (semantic) similarity.

(3) We introduce the framework (Section 4.3). We formalize the resource-bounded approximation scheme and establish the approximability theorem underlying the framework.

(4) We prove the approximability theorem for SPC (Section 4.4) and RA queries (Section 4.5), by developing resource-bounded algorithms with accuracy guarantees. The algorithms are based on search guided by access schema.

(5) We extend the study to RA extended with aggregate functions and group-by construct (Section 4.6). This makes the framework also capable of answering group-by aggregate queries.

(6) Using real-life and synthetic data, we experimentally validate the effectiveness of the resource-bounded framework. We find the following. (a) Our algorithms compute approximate answers with accuracy $\eta \geqslant 0.85$ for SPC queries, and 0.82 for RA queries, aggregate or not, when $\alpha \geqslant 5.5 \times 10^{-4}$ on all datasets, without making any assumptions on input queries. (b) They are able to find *exact answers* for many queries when α is as small as 2.6×10^{-6} for SPC and 4.1×10^{-6} for RA, reducing datasets of PB size to GB. (c) The algorithms are 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. (d) Our algorithms outperform sampling [AGPR99], histograms [IP99] and BlinkDB [AMP$^+$13] in accuracy for general SPC and RA queries. When α is 1.5×10^{-4}, its accuracy bound is 11.6, 3.7 and 2.0 times better than Sampl, Histo and BlinkDB, respectively.

We contend that the framework is promising for querying big data with bounded resources. In particular, access schema, accuracy measure and the resource-bounded approximation scheme yield practical tools for big data analysis.

4.1 Access Schema and α-Bounded Query Plans

In this section, we first extend access schema for approximation. We then study query plans under access schema to compute approximate answers within bounded resource.

4.1.1 Access Schema Extended for Approximation

Consider database schema $\mathcal{R}$ that consists of relation schemas $R_1, \ldots, R_n$. Each relation $R(A_1, \ldots, A_h)$ in $\mathcal{R}$ has attributes A_i with domain U_i for $i \in [1, h]$. Assume a function $\mathsf{dis}_{A_i} : U_i \times U_i \to \mathbb{R}$ to measure the distance between two A_i-attribute values, where $\mathbb{R}$ denotes real numbers.

For example, over relation $\mathsf{poi}(\mathsf{id}, \mathsf{type}, \mathsf{city}, \mathsf{street})$ of Example 13, $\mathsf{dis}_{\mathsf{type}}$ is defined in terms of semantic similarity, *e.g.,* $\mathsf{dis}_{\mathsf{type}}$(“*cinema*”, “*movie theater*”) is small; and $\mathsf{dis}_{\mathsf{street}}$ measures physical distance between two streets. A *trivial* distance function is 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.

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.

Recall access constraints in Chapter 2. Below we extend them to access templates.

Access templates. An *access template* over $\mathcal{R}$ has a form

$$\varphi(k) = R(X \to Y, k, \bar{d}_Y(k)),$$

where R, X and Y are the same as in access constraints, parameter k is a natural number, and $\bar{d}_Y[k]$ is a tuple with attributes Y of domain $\mathbb{R}$, called the *resolution tuple* of $\varphi(k)$.

An instance D of R *conforms to* $\varphi(k)$ if there is an index on X for Y such that for all $k \in (0, \lceil \log_2 |\pi_{XY}(D)| \rceil]$, given any X-value $\bar{a}$,

- it accesses and returns a set $\tilde{D}_Y^k(X = \bar{a})$ of at most 2^k distinct tuples in $D_Y(X = \bar{a})$, and
- for each tuple t in $D_Y(X = \bar{a})$, there exists a tuple t' in $\tilde{D}_Y^k(X = \bar{a})$ such that $|\mathsf{dis}_A(t[A], t'[A])| \leqslant \bar{d}_Y(k)[A]$ for each attribute $A \in Y$.

Intuitively, $\tilde{D}_Y^k(X = \bar{a})$ is an *abstraction* of $D_Y(X = \bar{a})$, with “resolution” (*i.e.,* “error” bounded by) $\bar{d}_Y(k)$. Observe that the index on D for $\varphi(k)$ is no larger than $2|D|$. Observe the following. (1) unlike access constraints, access templates impose no restriction on D. That is, any relation instance can conform to an access template by building index; and (2) the retrieved data and the amount of accessed data via access template is *dependent of* $|D|$ via the ratio α_φ, while that of access constraints is *independent of* $|D|$.

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

In this chapter, when it is clear from the context, an extended access schema is simply referred to as access schema as usual.

Example 14: Two access templates over $\mathcal{R}_0$ (Example 13) are

- $\varphi_1(k) = \mathsf{poi}(\mathsf{city} \to (\mathsf{id}, \mathsf{type}, \mathsf{street}), k, \bar{d}_{\mathsf{street}}(k))$, and
- $\varphi_2(k) = \mathsf{poi}(\emptyset \to (\mathsf{id}, \mathsf{type}, \mathsf{city}, \mathsf{street}), k, \bar{d}_{\mathsf{poi}}(k))$.

Here φ_1 gives us an index that for any city, returns at most 2^k “representative triples” of (id, type, street), along with a resolution tuple $\bar{d}_{\mathsf{street}}(k)$ to ensure the accuracy of these triples. Template $\varphi_2(k)$ fetches 2^k "representative tuples" of poi for a given k value, with accuracy bounded by $\bar{d}_{\mathsf{poi}}(k)$.

An access schema $\mathcal{A}_1$ over $\mathcal{R}_0$ is the set consisting of templates $\varphi_1(k)$ and $\varphi_2(k)$, and constraints ψ_1, ψ_2 of $\mathcal{A}_c^1$. □

4.1.2 α-Bounded Query Plans under Access Schema

Access schema $\mathcal{A}$ over $\mathcal{R}$ allows us to control access to data of $\mathcal{R}$ and respect resource ratio α. Consider an RA query Q defined over $\mathcal{R}$ with selection σ, projection π, Cartesian product $\times$, union $\cup$, set difference $-$ and renaming ρ.

Query plans. We extend query plans of Chapter 2 to incorporate access templates and the resource ratio. More specifically, we consider query plans of the form:

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

where for $i \in [1,n]$, 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 $R(X \to Y, N)$ in $\mathcal{A}$, or an access template $R(X \to Y, k, \bar{d}_Y(k))$ in $\mathcal{A}$; the fetch operation retrieves a set W from D, where W is $D_{XY}(X = \bar{a})$ if φ is a constraint, and is $\tilde{D}_Y^k(X = \bar{a})$ if φ is a template; it returns $\bigcup_{\bar{a} \in T_j(D)} \{(\bar{a}, \bar{b}) \mid \bar{b} \in W\}$.

An *execution plan* ξ_* of ξ in a database D is an instantiation of ξ such that for each k in a fetch operation via template $R(X \to Y, k, \bar{d}_Y(k))$, k is a value in $(0, \lceil \log_2 |\pi_{XY}(D)| \rceil]$.

Intuitively, ξ_* executes the operations δ_i one by one from $i = 1$ to n as in Chapter 2, except that data can only be fetched via the indices in the constraints or templates of $\mathcal{A}$. We denote by $\xi_*(D)$ the answers to Q in D computed by ξ_*.

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 of $\mathcal{R}$, if $D \models \mathcal{A}$, then $\xi_*^m(D) = Q(D)$, where ξ_*^m is the execution plan of ξ in which for each template $R(X \to Y, k, \bar{d}_Y(k))$, $k = \lceil \log_2 |\pi_{XY}(D)| \rceil$ and $\bar{d}_Y(k) = (0, \ldots, 0)$, *i.e.,* without "errors". That is, when ξ is allowed to fetch the entire dataset D, it computes the exact answers $Q(D)$.

α-bounded execution plans. The *data tariff* of an execution plan ξ_* for Q in D is the sum of the N's in the constraints and 2^k's ($k \leqslant \lceil \log_2 |\pi_{XY}(D)| \rceil$) in the templates used in ξ_*. If ξ_* has tariff M, it accesses at most M tuples in D in the entire process, as it accesses D only via indices for $\mathcal{A}$.

symbols	notations
$\mathcal{R}$, R	database schema $\mathcal{R}$ and $R \in \mathcal{R}$
α, η	resource ratio and accuracy bound
$\mathcal{A}$	access schema (constraints, templates)
ψ	access constraint $R(X \to Y, N)$
$\varphi(k)$	access templates $R(X \to Y, k, \bar{d}_Y(k))$
ξ (ξ_*)	query (execution) plan with fetch operations
$\delta_{\mathsf{rel}}(Q,s)$, $\delta_{\mathsf{cov}}(Q,t,s)$	distance functions (relevance, coverage)
$\mathsf{F}_{\mathsf{rel}}(S,Q,D)$, $\mathsf{F}_{\mathsf{cov}}(S,Q,D)$	the relevance and coverage of S
$S = \Gamma(Q,D,\alpha)$	approximate answers, by Γ *w.r.t.* α
$\mathsf{accuracy}(S,Q,D)$	$\mathsf{min}(\mathsf{F}_{\mathsf{rel}}(S,Q,D), \mathsf{F}_{\mathsf{cov}}(S,Q,D))$

Table 4.1: Notations in Chapter 4

We say that ξ_* is α-*bounded in D* if its tariff in D is at most $\alpha|D|$. We consider datasets D of at least millions of tuples.

Example 15: Given a resource ratio α, query Q_2 of Example 13 can be answered under $\mathcal{A}_1$ of Example 14 as follows: (1) identify all city values by using the plan for Q_1 given in Example 13; (2) for each city value above, fetch $(\mathsf{id}, \mathsf{type}, \mathsf{street})$ of poi *w.r.t.* α, by using φ_1 in $\mathcal{A}_1$; and (3) compute approximate answers in the fetched data, such that type is *close* to "cinema". A formal query plan will be given in Section 4.4. $\square$

Note that a query Q is boundedly evaluable (recall the definition of boundedly evaluable queries in Chapter 2) if it has a query plan that uses access constraints of $\mathcal{A}$ only, without using templates. In other words, boundedly evaluable query plans of Chapter 2 are a special case of query plans here, which compute exact answers $Q(D)$ and have a fixed data tariff regardless of α and independent of the size $|D|$ of underlying dataset D.

The notations of this chapter are summarized in Table 4.1.

4.2 Accuracy of Approximate Answers

Consider an algorithm Γ that, given a resource ratio $\alpha \in (0,1]$, an RA query Q and a database D, computes a set S of approximate answers, denoted as $\Gamma(Q,D,\alpha)$, by accessing at most an α-fraction of D. We want to assess how accurate S is and thus, how “good” algorithm Γ is.

Previous accuracy metrics typically compare S and $Q(D)$, or assess the “losslessness” of a synopsis of D. They do not work very well on resource-bounded approximation. Consider, for instance, F-measure, defined as $F(S,Q,D) = 2\,\frac{\mathsf{precs}(S,Q,D)\ \mathsf{recall}(S,Q,D)}{\mathsf{precs}(S,Q,D)+\mathsf{recall}(S,Q,D)}$, where $\mathsf{precs}(S,Q,D) = \frac{|S \cap Q(D)|}{|S|}$ and $\mathsf{recall}(S,Q,D) = \frac{|S \cap Q(D)|}{|Q(D)|}$. For the majority of relational queries Q, without knowing D, it is not possible to produce answers S such that $S \cap Q(D) \neq \emptyset$ (take *e.g.,* data selection query $\sigma_{A=B}R(A,B)$ for example). In other words, there exists a dataset D such that $F(S,Q,D) = \emptyset$ where $S = \Gamma(Q,D,\alpha)$ for deterministic algorithm Γ that evaluates Q with resource ratio $\alpha < 1$. That is, based on F-measure, no approximation algorithm is “good”, since for the majority of RA queries Q, $\mathsf{min}_D F(S,Q,D) = 0$, where S is returned by the algorithm.

However, the situation is not so hopeless. Below we define two distance functions (Section 4.2.1), and introduce a measure with the functions (Section 4.2.2).

4.2.1 Approximation Distance

For a set S of approximate answers to Q in D, we measure its accuracy with two *distance functions*:

- $\delta_{\mathsf{rel}}(Q,s)$ assesses how *relevant* an approximate answer $s \in S$ is to query Q in D, and
- $\delta_{\mathsf{cov}}(Q,t,s)$ denotes the differences between an exact answer $t \in Q(D)$ and an approximate answer $s \in S$.

Below we inductively define $\delta_{\mathsf{rel}}(Q,s)$ and $\delta_{\mathsf{cov}}(Q,t,s)$ based on the structure of Q, in terms of *normalized* distance functions dis_A across difference domains of attributes.

(1) When Q is R (similarly for $\rho(R)$).

- $\delta_{\mathsf{rel}}(Q,s) = 0$ if s is a tuple in the instance of R in D, and $\delta_{\mathsf{rel}}(Q,s) = +\infty$ otherwise;
- $\delta_{\mathsf{cov}}(Q,t,s) = \sum_{A \in R_Q} w_A \cdot |\mathsf{dis}_A(t[A], s[A])|$;

where w_A is “weight” for attribute A (optionally) provided by the users, indicating its relevant importance to them.

Intuitively, (1) if s is a tuple in D, then s is relevant to Q; otherwise s is irrelevant; and (2) the distance between exact answer t and approximate answer s is measured as the *sum* of the differences in all their attributes A in terms of dis_A.

(2) When Q is $\sigma_{A=c}(Q')$.

- $\delta_{\mathsf{rel}}(Q,s) = \delta_{\mathsf{rel}}(Q',s) + |\mathsf{dis}_A(c,s[A])|$.
- $\delta_{\mathsf{cov}}(Q,t,s) = \delta_{\mathsf{cov}}(Q',t,s)$.

Intuitively, (1) s is relevant to Q if (a) s is relevant to sub-query Q' and (b) $s[A]$ is close to constant c in the selection condition of Q; and (2) the distance between t and s *w.r.t.* Q is measured as the distance between t and s *w.r.t.* Q'.

(3) When Q is $Q_1 \times Q_2$. Let $t = (t_1,t_2)$ and $s = (s_1,s_2)$, where t_i (resp. s_i) is an exact answer (resp. approximate answer derived from s) to Q_i for $i \in [1,2]$. Then

- $\delta_{\mathsf{rel}}(Q,s) = \delta_{\mathsf{rel}}(Q_1,s_1) + \delta_{\mathsf{rel}}(Q_2,s_2))$, and
- $\delta_{\mathsf{cov}}(Q,t,s) = \delta_{\mathsf{cov}}(Q_1,t_1,s_1) + \delta_{\mathsf{cov}}(Q_2,t_2,s_2))$.

That is, (1) s is relevant to Q if s_1 and s_2 are relevant to Q_1 and Q_2, respectively; and (2) the distance between exact answer t and approximate answer s *w.r.t.* Q is determined by the distance between t_i and s_i *w.r.t.* Q_i for $i \in [1,2]$.

(4) When Q is $\sigma_{A \leqslant c}(Q')$.

- $\delta_{\mathsf{rel}}(Q,s) = \delta_{\mathsf{rel}}(Q',s)$ if $\mathsf{dis}_A(c,s[A]) \leqslant 0$, and $\mathsf{dist}(Q',s) + \mathsf{dis}_A(c,s[A])$ otherwise;
- $\delta_{\mathsf{cov}}(Q,t,s)$ is the same as its counterpart for $\sigma_{A=\mathrm{c}}(Q')$;

similarly for $\sigma_{A \geqslant \mathrm{c}}(Q')$.

(5) When Q is $\sigma_{A=B}(Q')$.

- $\delta_{\mathsf{rel}}(Q,s) = \delta_{\mathsf{rel}}(Q',s) + |\mathsf{dis}_A(s[A],s[B])|$.
- $\delta_{\mathsf{cov}}(Q,t,s) = \delta_{\mathsf{cov}}(Q',t,s)$.

Similar to (2), s is relevant to Q if it is relevant to Q' and $s[A]$ is close to $s[B]$ as specified by the selection condition.

(6) When Q is $\sigma_{A \leqslant B}(Q')$.

- $\delta_{\mathsf{rel}}(Q,s) = \delta_{\mathsf{rel}}(Q',s) + \max(0, \mathsf{dis}_A(s[B],s[A]))$.
- $\delta_{\mathsf{cov}}(Q,t,s) = \delta_{\mathsf{cov}}(Q',t,s)$.

Similar to (5), s is relevant to Q if it is relevant to Q' and $s[A]$ is smaller or not too larger than $S[B]$.

(7) When Q is $\pi_Y(Q')$. Denote by s' a tuple of relation $R_{Q'}$ that draws values from the active domain of D such that $\pi_Y(s') = s$, then

- $\delta_{\mathsf{rel}}(Q,s) = \min_{s'} \delta_{\mathsf{rel}}(Q',s')$.
- $\delta_{\mathsf{cov}}(Q,t,s) = \sum_{A \in Y} |\mathsf{dis}_A(t[A],s[A])|$.

That is, (1) $\delta_{\mathsf{rel}}(Q,s)$ represents the distance of the most relevant s' to Q' among all approximate answers to Q' such that $\pi_Y(s') = s$; and (2) $\delta_{\mathsf{cov}}(Q,t,s)$ is defined along the same lines as (1), *i.e.,* determined by the sum of the differences between t and s between their attributes A in terms of dis_A.

(8) When Q is $Q_1 \cup Q_2$.

- $\delta_{\mathsf{rel}}(Q,s) = \mathsf{min}(\delta_{\mathsf{rel}}(Q_1,s), \delta_{\mathsf{rel}}(Q_2,s))$.
- $\delta_{\mathsf{cov}}(Q,t,s) = \delta_{\mathsf{cov}}(Q_1,t,s)$ if t is in $Q_1(D)$, and $\delta_{\mathsf{cov}}(Q,t,s) = \delta_{\mathsf{cov}}(Q_2,t,s)$ otherwise.

That is, (1) s is relevant to Q if s is relevant to either Q_1 or Q_2; and (2) the distance between exact answer t and approximate answer s *w.r.t.* Q is either $\delta_{\mathsf{cov}}(Q_1,t,s)$ or $\delta_{\mathsf{cov}}(Q_2,t,s)$. When $t \in Q_1(D) \cap Q_2(D)$, $\delta_{\mathsf{cov}}(Q_1,t,s) = \delta_{\mathsf{cov}}(Q_2,t,s)$.

(9) When Q is $Q_1 - Q_2$.

- $\delta_{\mathsf{rel}}(Q,s)$ is $\delta_{\mathsf{rel}}(Q_1,s)$ if $s \notin Q_2(D)$, and $+\infty$ otherwise.
- $\delta_{\mathsf{cov}}(Q,t,s) = \delta_{\mathsf{cov}}(Q_1,t,s)$.

Intuitively, (1) when s is in $Q_2(D)$, s cannot be an approximate answer to Q; that is, we enforce the set-difference semantics via $\delta_{\mathsf{rel}}(Q,s)$. (2) The coverage distance $\delta_{\mathsf{cov}}(Q_1,t,s)$ carries over to $\delta_{\mathsf{cov}}(Q_1,t,s)$, as $t \in Q_1(D)$ when $t \in Q(D)$.

Observe the following. (1) The distance functions "approximate" selection conditions only. (2) For any *exact answer $t \in Q(D)$*, $\delta_{\mathsf{rel}}(Q,t) = 0$. *i.e.,* most relevant. (3) How $\delta_{\mathsf{rel}}(Q,s)$ is computed *explains* why s is an approximate answer.

4.2.2 Accuracy Measure

Based on $\delta_{\mathsf{rel}}(Q,s)$ and $\delta_{\mathsf{cov}}(Q,t,s)$, we now define a measure for a set S of approximate answers to Q in D, referred to as the *the C-measure*. It is characterized by the *relevance ratio* $\mathsf{F}_{\mathsf{rel}}(\xi(D),Q,D)$ *of S to Q in D*, and the *coverage ratio* $\mathsf{F}_{\mathsf{cov}}(S,Q,D)$ *of $Q(D)$ by S*, respectively, given as follows:

$$\mathsf{F}_{\mathsf{rel}}(S,Q,D) = \frac{1}{1+\mathsf{max}_{s\in S}\delta_{\mathsf{rel}}(Q,s)}, \tag{4.1}$$

$$\mathsf{F}_{\mathsf{cov}}(S,Q,D) = \frac{1}{1+\mathsf{max}_{t\in Q(D)}\mathsf{min}_{s\in S}\delta_{\mathsf{cov}}(Q,t,s)} \tag{4.2}$$

Intuitively, (1) $\mathsf{F}_{\mathsf{rel}}(S,Q,D)$ indicates how "sensible" tuples in S are as approximate answers to Q, and $\mathsf{F}_{\mathsf{cov}}(S,Q,D)$ indicates how well S "covers" exact answers $Q(D)$. (2) The accuracy is *deterministic*: each and every approximate answer $s \in S$ is both relevant and close to an exact answer. (3) 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. (4) Both $\mathsf{F}_{\mathsf{rel}}(S,Q,D)$ and

$\mathsf{F}_{\mathsf{cov}}(S,Q,D)$ are 1 when $S = Q(D)$. In particular, we define $\mathsf{F}_{\mathsf{cov}}(S,Q,D) = 1$ for any S if $Q(D) = \emptyset$; and $\mathsf{F}_{\mathsf{cov}}(S,Q,D) = 0$ if $S = \emptyset$ and $Q(D) \neq \emptyset$.

For example, when executing the query plan of Q_2 under $\mathcal{A}_2$ (Example 15) in a database D_0 of 200 million tuples, we set $k = 14$ in the fetch operation with template φ_1, for $\alpha = 10^{-4}$. The plan fetches 16384 $(\mathsf{poi}, \mathsf{type}, \mathsf{street})$ tuples from poi with $\bar{d}_{\mathsf{street}}$ = $(+\infty$, 0.1, 0.1) (normalized distance). It finds approximate answers with relevance and coverage above $\frac{1}{1+0.1}$ = 0.91, by accessing 16384 tuples in total.

4.3 Resource Bounded Query Answering

Based on α-bounded query plans and the C-measure, we now present the resource-bounded approximation scheme and the resource-bounded framework for relational queries.

4.3.1 Resource Bounded Approximation

Assume an access schema $\mathcal{A}$ over a database schema $\mathcal{R}$.

Scheme. A *resource-bounded approximation scheme* under $\mathcal{A}$ is an algorithm $\Gamma_{\mathcal{A}}$ such that for *any* query RA query Q and *any* resource ratio $\alpha \in (0,1]$, it generates

- a query plan ξ for Q under $\mathcal{A}$, and
- for any database $D \models \mathcal{A}$ of $\mathcal{R}$, an α-bounded execution plan ξ_* of ξ in D with an C bound η such that $\mathsf{F}_{\mathsf{rel}}(\xi_*(D),Q,D) \geqslant \eta$ and $\mathsf{F}_{\mathsf{cov}}(\xi_*(D),Q,D) \geqslant \eta$.

Observe the following about the approximation scheme.

(1) It generates ξ guided by Q with $\mathcal{A}$ such that *for all* databases D of $\mathcal{R}$, it has an execution plan ξ_* in D such that (a) ξ_* computes approximate answers $\xi_*(D)$ by accessing at most $\alpha|D|$ tuples, and (b) ξ_* guarantees relevance and coverage of each answer to be at least η.

(2) It takes resource ratio α as a parameter, allowing us to query big D with bounded resources by setting α small. Here α denotes the "resolution" of the data that we can afford: the higher the resolution is, the more accurate $\xi_*(D)$ is.

(3) As shown in Fig. 1.1, it is based on *dynamic data reduction*: for each input Q, it generates (different) α-bounded plans ξ_* to pursue the highest accuracy under the constraint of resource ratio α, as opposed to one-size-fit-all synopsis.

Approximability. Is there always an access schema $\mathcal{A}$ that yields a resource-bounded approximation scheme?

As shown in Chapter 2, a set $\mathcal{A}_c$ of access constraints over $\mathcal{R}$ can be automatically discovered and helps us boundedly evaluate a number of RA queries. For an instance D of $\mathcal{R}$, denote by $I_{\mathcal{A}_c}$ and $\|\mathcal{A}_c\|$ the indices and number of all constraints in $\mathcal{A}_c$, respectively. Then $I_{\mathcal{A}_c}$ is at most $O(\|\mathcal{A}_c\||D|)$ [CF16].

We can always extend $\mathcal{A}_c$ to an access schema $\mathcal{A}$ for approximation. Denote by $|\mathcal{R}|$ the total number of attributes in $\mathcal{R}$. An RA query is called *quantitative* if all its selection conditions or projections are defined on attributes A of a domain with a nontrivial distance function dis_A (see Section 4.1).

Theorem 29: *Given any set $\mathcal{A}_c$ of access constraints over a database schema $\mathcal{R}$, there exists a set $\mathcal{A}_t$ of access templates such that for any instance D of $\mathcal{R}$, if $D \models \mathcal{A}_c$ then*

(1) $D \models \mathcal{A}_0$, where $\mathcal{A}_0 = \mathcal{A}_t \cup \mathcal{A}_t$;

(2) $\|\mathcal{A}_t\| \leqslant |\mathcal{R}| + \|\mathcal{A}_c\|$ and $I_{\mathcal{A}}$ is in $O(\|\mathcal{A}_0\||D|)$; and

(3) there is an approximation scheme $\Gamma_{\mathcal{A}}$ such that for all resource ratios $\alpha \in (0,1]$ and all RA *queries Q over $\mathcal{R}$, it generates an α-bounded plan ξ_* with C bound η; moreover, $\eta > 0$ if Q is a quantitative query and*

(a) Q is in SPC *(selection, projection, product), or*

(b) Q is in RA *and $\xi_*(D) \neq \emptyset$.* □

That is, we can always extend $\mathcal{A}_c$ to an access schema $\mathcal{A}_0$ does not substantially increase the cost of indexing, and can answer *all* RA queries Q over $\mathcal{R}$ with any α bounded resources in instances D of $\mathcal{R}$ that satisfy $\mathcal{A}_c$. Better yet, when Q is quantitative, it has a provable accuracy $\eta > 0$. As will be seen in Section 4.7, $\eta \geqslant 0.9$ when $\alpha = 5.5 \times 10^{-4}$; on average $I_{\mathcal{A}_c}$ accounts for 0.07% of $|D|$, and $I_{\mathcal{A}_0}$ in the range of typical database indices [Lev].

Below we sketch a proof for Theorem 29(1-2) for details). We prove Theorem 29 (3a) and (3b) in Sections 4.4 and 4.5, respectively, by providing algorithms for generating α-bounded plans for SPC and RA, respectively.

Proof of Theorem 29(1-2). We define $\mathcal{A}_t$ as follows: (a) for each schema R of $\mathcal{R}$, include template $\varphi_R(k) = R(\emptyset \to \mathsf{attr}(R), k, \bar{d}_R(k))$ in $\mathcal{A}_t$, where $\mathsf{attr}(R)$ is the set of all attributes of R; (b) for each access constraint $R(X \to Y, N)$ in $\mathcal{A}_c$, include $R(XY \to \mathsf{attr}(R) \setminus XY, k, \bar{d}(k))$; and (c) for each attribute A in some schema R of $\mathcal{R}$, if there exists a constraint $R'(X' \to Y', N)$ in $\mathcal{A}_c$ such that $R'[A']$ and $R[A]$ have the same domain and $A' \in Y'$, then include $R(A \to \mathsf{attr}(R) \setminus \{A\}, k, \bar{d}(k))$. As will be seen in Sections 4.4

and 4.5, $\mathcal{A}_0 = \mathcal{A}_t \cup \mathcal{A}_c$ suffices for resource-bounded approximation. Obviously, $\|\mathcal{A}_t\| = |\mathcal{R}| + \|\mathcal{A}_c\|$. For any instance D of $\mathcal{R}$, the size of the index for $R(X \to Y, k, \bar{d}_R(k))$ is at most $2|D_R|$, where D_R is the instance of R in D. Thus the size of $I_{\mathcal{A}_t}$ is in $2|D|\|\mathcal{A}_t\|$, and $I_{\mathcal{A}_0}$ remains in $O(|D|\|\mathcal{A}_0\|)$ (recall the index size analysis in Chapter 3). □

For example, access schema $\mathcal{A}_2$ of Example 14 extends $\mathcal{A}_c^1$ of Example 13 by including type (a) and (c) templates in the proof of Theorem 29(1-2) above.

4.3.2 A Resource Bounded Framework

We next present our framework for answering relational queries with bounded resources, by combining bounded evaluation in Chapter 3 and resource-bounded approximation. It is to be built on top of DBMS, and to extend existing DBMS with a functionality of resource-bounded query evaluation.

As shown in Fig. 4.2, the framework consists of two parts.

(1) Offline algorithms. For an application with databases D of schema $\mathcal{R}$, it works as follows. As preprocessing (**C1**), it discovers a set $\mathcal{A}_c$ of access constraints following Chapter 3, and extends it to an access schema $\mathcal{A}_0$ with templates as described above. It builds indices $I_{\mathcal{A}_0}$ for $\mathcal{A}_0$ on D, and maintains $I_{\mathcal{A}_0}$ in response to updates to D (**C2**; see details shortly). It also determines resource ratio α based on available resources, workload of the application and the size of D.

(2) Online algorithms. For any query Q posed on D, the framework first checks whether Q is boundedly evaluable under the access constraints of $\mathcal{A}_c$ (**C3**). This is carried out efficiently based on an effective syntax for boundedly evaluable RA queries (cf. Chapter 3). If so, it generates a boundedly evaluable query plan ξ (**C4**) and executes ξ directly using the underlying DBMS by accessing a bounded dataset $D_Q \subseteq D$ (**C5**). The algorithms for **C3–C5** have been developed in Chapter 3.

If Q is not bounded, the framework invokes a resource-bounded approximation scheme $\Gamma_{\mathcal{A}_0}$ such that for any database D of $\mathcal{R}$, if $D \models \mathcal{A}_c$, then it generates an α-bounded execution plan ξ_*, computes an accuracy bound η (Theorem 29, **C6**), and executes ξ_* by the DBMS; it accesses no more than $\alpha|D|$ tuples in the entire process. It returns $(Q_{\xi_*}(D), \eta)$ (**C7**). The algorithms for **C6** will be provided in Sections 4.4 and 4.5.

As will be seen in Section 4.6, the framework can be readily extended to evaluating RA extended with aggregate functions. That is, the framework is able to answer relational queries, aggregate or not, with bounded resources.

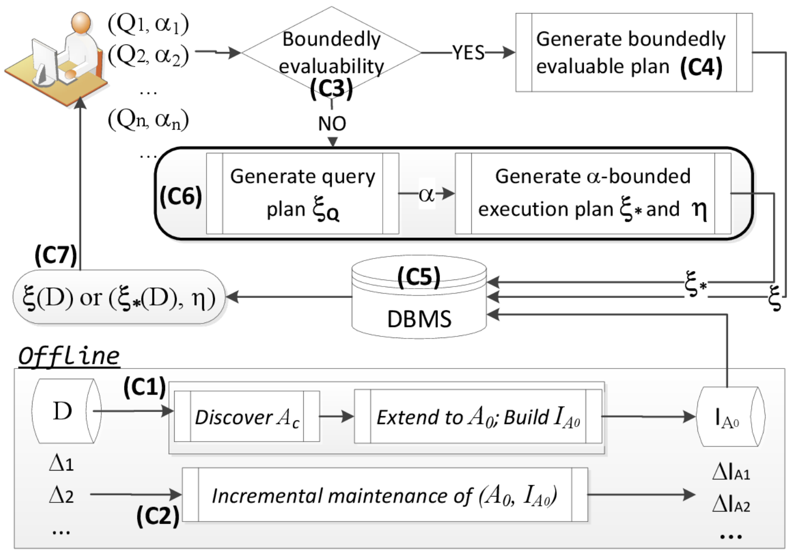


Figure 4.2: Resource-bounded framework

Example 16: Under access schema $\mathcal{A}_2$ of Example 14, the framework answers Q_1 and Q_2 of Example 13 as follows.

(1) It finds that Q_1 is boundedly evaluable under $\mathcal{A}_2$ (**C3**), and thus generates a bounded query plan ξ for Q (**C4**), which is excuted by DBMS (**C5**). It returns exact answers $\xi(D)$.

(2) It determines that Q_2 is not boundedly evaluable under $\mathcal{A}_2$ (**C3**). Thus it generates an α-bounded execution plan ξ_* for Q_2 under $\mathcal{A}_2$ for a given α, along with an accuracy bound η (**C6**). The plan is carried out by DBMS, and returns approximate answers $\xi_*(D)$ and accuracy bound η (**C7**). □

Index maintenance. When D is updated with ΔD, *i.e.,* tuple insertions and deletions (which can simulate value modifications), indices in $I_{\mathcal{A}_0}$ can be maintained by a *bounded incremental* algorithm [RR96]. Its cost is determined by $\mathcal{A}_0$ and size $|\Delta D|$ only, not by $|D|$ and $|I_{\mathcal{A}_0}|$. Such algorithms are efficient when ΔD is small, as commonly found in practice [RR96]. As observed in [RR96], such an incremental algorithms is efficient when ΔD is small as commonly found in practice.

Proposition 30: *For any changes ΔD to D, $I_{\mathcal{A}_0}$ can be updated in $O(N|\Delta D|)$ time, where $N = \Sigma_{R(X\to Y,N)\in\mathcal{A}'_c} N + \Sigma_{\varphi\in\mathcal{A}'_t} k$, where $\mathcal{A}'_c$ (resp. $\mathcal{A}'_t$) contains constraints (resp. templates) of $\mathcal{A}_0$ defined on relations that ΔD updates.* □

Proof: Observe first that for each access template $\varphi \in \mathcal{A}'_t$, where $\varphi = R(X \to$

$Y, k', \bar{d}_Y(k'))$ and parameter k' can be instantiated to any value in $[0,k]$, the index for φ consists of $k+1$ sets of tuples of size 2^0, 2^1, …, 2^k, respectively (recall that $k = \lceil \log_2 |\pi_{XY}(D)| \rceil$) *i.e.,* the total size of the index for φ is $\sum_{i=0}^{k} = 2^{k+1} - 1$.

Given ΔD, we update the $(2^{k+1}-1)$-size index for φ as follows. For each tuple $t \in \Delta D$, and for each instantiation value v in $[0,k]$ for k', we randomly pick 1 tuple from the 2^v-size index by accessing 1 tuple, and check whether $\bar{d}_Y(v)$ needs to be changed when t is merged with s (*i.e.,* we either replace s with t or discard t and use s as a representative tuple instead). This takes $O(|\Delta D|(k+1)) = O(k|\Delta D|)$ time to update index for φ. Therefore, the total time for updating all indices for $\mathcal{A}_t$ is at most $|\Delta D| \cdot \sum_{\varphi \in \mathcal{A}'_t} k$, where $\mathcal{A}'_t$ contains templates of $\mathcal{A}$ on relation schemas over which ΔD is defined, and k is the *maximum number* that the parameter of φ can be instantiated. Putting these together with the result of Chapter 3 for $\mathcal{A}_c$, we get Proposition 30. □

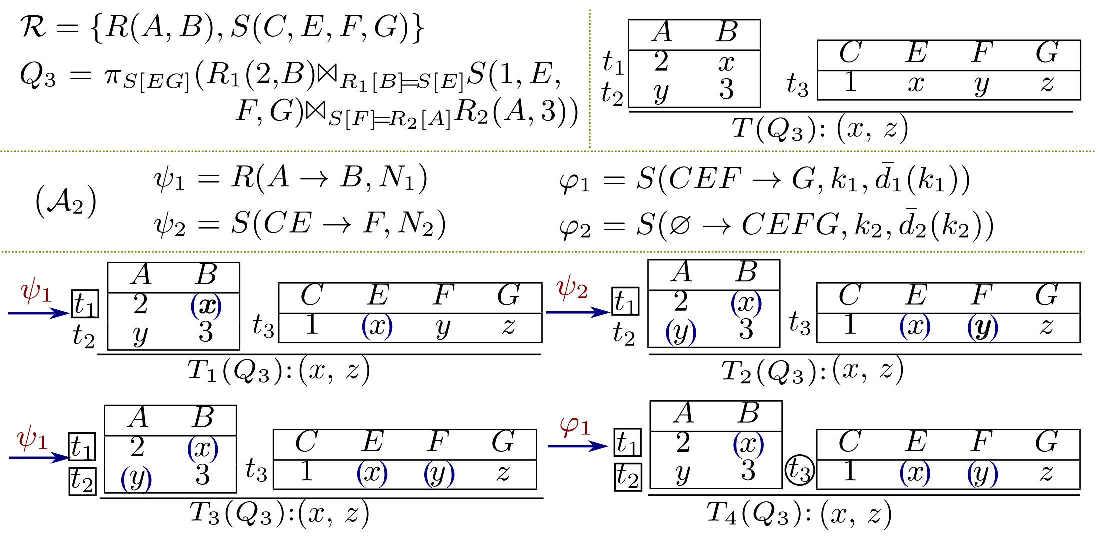
Remark. The algorithm is optimal, since it is absolutely necessary for any incremental maintenance algorithm to examine every layer of the index (*i.e.,* index for each instantiation of the parameter) of each involved access constraint or template, which is precisely the cost incurred by the algorithm. While there are other incremental algorithms that can update the indices of $\mathcal{A}$ with higher accuracy on the resolution tuples, those algorithms incur higher complexity as a price.

4.4 Approximating SPC Queries

As a proof of Theorem 29(3a), we develop a resource-bounded approximation scheme for SPC under the access schema $\mathcal{A}_0$ given in the proof of Theorem 29(1-2). It consists of two algorithms: (1) parQP$_{\mathsf{SPC}}$ that, given an SPC query Q, generates a query plan ξ_Q for Q under $\mathcal{A}_0$ (Section 4.4.1); and (2) instQP$_{\mathsf{SPC}}$ that, given a resource ratio α and a database $D \models \mathcal{A}_0$, instantiates ξ_Q to get an α-bounded execution plan for Q in D (Section 4.4.2).

4.4.1 Generating Canonical Query Plans

Algorithm parQP$_{\mathsf{SPC}}$ is based on the following notion. Under $\mathcal{A}_0$, a *canonical query plan* ξ_Q for an SPC query Q is of the form (ξ_Q^F, ξ_Q^E), where (a) ξ_Q^F is a *fetching plan* for Q under $\mathcal{A}_0$, which is a sequence of fetch operations connected by projection and product, to fetch attribute values needed for Q; and (b) ξ_Q^E is an *evaluation plan* for Q, which performs the relational operations of Q using the data fetched by ξ_Q^F. That is, ξ_Q first



$\mathcal{R} = \{R(A,B), S(C,E,F,G)\}$
$Q_3 = \pi_{S[EG]}(R_1(2,B) \bowtie_{R_1[B] \models S[E]} S(1,E,F,G) \bowtie_{S[F] \models R_2[A]} R_2(A,3))$
$T(Q_3)$: (x, z)
$(\mathcal{A}_2)$
$\psi_1 = R(A \to B, N_1)$
$\psi_2 = S(CE \to F, N_2)$
$\varphi_1 = S(CEF \to G, k_1, \bar{d}_1(k_1))$
$\varphi_2 = S(\emptyset \to CEFG, k_2, \bar{d}_2(k_2))$
$T_1(Q_3)$: (x, z)
$T_2(Q_3)$: (x, z)
$T_3(Q_3)$: (x, z)
$T_4(Q_3)$: (x, z)

Figure 4.3: A chasing sequence for Q_3 under $\mathcal{A}_2$

fetches necessary data D_Q, and then computes (approximate) answers to Q using D_Q. Such plans are a "normal form".

Lemma 31: *Under the access schema $\mathcal{A}_0$ of Theorem 29, every* SPC *query Q has a canonical bounded query plan.* □

Proof: It is obvious that access templates of type (a) given in the proof of Theorem 29(1-2) suffices to prove Lemma 31. Indeed, every SPC query has a bounded query plan that fetches all relations in Q via the templates first, and then evaluates Q directly using the fetched data. □

By Lemma 31, parQP$_{\mathsf{SPC}}$ only needs to generate canonical plans. The tricky part is fetching plan ξ_Q^F, to identify what attributes are needed to answer Q and how to retrieve the data via the indices embedded in access schema $\mathcal{A}_0$.

Fetching plan. To find ξ_Q^F, we use *chasing*, a classical technique [AHV95] in dependency theory, such that each chasing step corresponds to a fetch operation with an access constraint or an access template in $\mathcal{A}_0$. The chasing is defined on the tableau representation of SPC queries.

The *tableau* of an SPC Q is a pair $(T(Q), u(Q))$, where (a) $T(Q)$ is a collection of tables in which tuples represent relation atoms in Q; and (b) $u(Q)$ is a tuple of variables specifying the output of Q (cf. [AHV95]). For example, an SPC query Q_3 and its tableau are at the top of Fig. 4.3.

Intuitively, a tuple in $T(Q)$ is a template with variables to be mapped to attribute values, and $u(Q)$ denotes projected attributes. Computing $Q(D)$ is essentially to fetch tuples in D that match templates in $T(Q)$, and instantiate $u(Q)$. We refer to $T(Q)$ as the tableau of Q when it is clear in the context.

Chase. A *chasing sequence* for Q with an access schema $\mathcal{A}$ 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 (a) $T_i(Q)$ is $T(Q)$ in which some tuples are marked *exactly* or *approximately* covered (enclosed in square or circle in Fig. 4.3), respectively, and some variables are marked *covered* (enclosed in parenthesis); in particular, $T_0(Q)$ is $T(Q)$ without annotations, and (b) γ_i is a constraint or a template in $\mathcal{A}$ that is applied to $T_{i-1}(Q)$ and triggers the marking.

Intuitively, each $T_i(Q)$ indicates a fetch operation that retrieves attribute values to instantiate variables in $T(Q)$, using an access constraint or access template in $\mathcal{A}$.

The chasing starts with $T(Q)$, and applies constraints or templates in each chasing step, until no more tuples can be marked. Each chasing step $T_i(Q) \xmapsto{\gamma_i} T_{i+1}(Q)$ identifies what variables can be "instantiated" and what tuples can be fetched via $\gamma_i \in \mathcal{A}$. It applies one of the following rules.

(1) A *variable* y in a tuple t of $T_{i+1}(Q)$ is marked *covered* in $T_{i+1}(Q)$ if (a) y is not yet covered in $T_i(Q)$ and (b) there exists a constraint $R(X \to Y, N)$ in $\mathcal{A}$ such that t is a tuple of (a renaming of) R in which $t[X]$ consists of constants or covered variables in $T_i(Q)$, and y is in $t[Y]$.

(2) A *tuple* t of R in $T(Q)$ is marked *exactly covered* in $T_{i+1}(Q)$ if (a) t is not exactly covered in $T_i(Q)$; and (b) there is a constraint $R(X \to Y, N)$ in $\mathcal{A}$ such that $t[X]$ consists of constants or covered variables in $T_i(Q)$, and $t[XY]$ contains all constants and nontrivial variables and constants in t. A variable is *nontrivial* if it appears in $u(Q)$ or occurs multiple times in $T(Q)$.

(3) *Tuple* t of R in $T(Q)$ is marked *approximately covered* in $T_{i+1}(Q)$ if (a) t is not covered in $T_i(Q)$, exactly or approximately; and (b) there is a template $R(X \to Y, k, \bar{d}(k))$ such that $t[X]$ consists of constants and covered variables in $T_i(Q)$, and $t[XY]$ has all nontrivial variables and constants in t.

Each tuple has at most one annotation; if it is exactly covered, it will be not be marked approximately covered.

For example, Figure 4.3 depicts a chasing sequence of 4 steps for Q_3 under $\mathcal{A}_3$ over $T(Q_3)$. It marks variables x and y covered (marked in parenthesis); tuples t_1 and t_2 exactly covered (in square), and t_3 approximately covered (in circle).

The chasing procedure for Q with $\mathcal{A}$, denoted by $\mathsf{Chase}(Q,\mathcal{A})$, applies the rules to $T(Q)$, and generates chasing sequences. We say that $\mathsf{Chase}(Q, \mathcal{A})$ is *terminal* if it always terminates. It has the *Church-Rosser* property if all chasing sequences for Q with $\mathcal{A}$ terminate at the same annotated $T_m(Q)$, referred to as *the output* of $\mathsf{Chase}(Q, \mathcal{A})$.

Lemma 32: *(1) For any* SPC *query Q and any access schema $\mathcal{A}$, the chasing* $\mathsf{Chase}(Q,\mathcal{A})$ *terminates in $|Q| + 2\|T(Q)\|$ steps and has the Church-Rosser property. (2) With the access schema $\mathcal{A}_0$ of Theorem 29, all tuples in the output of* $\mathsf{Chase}(Q,\mathcal{A})$ *are covered, exactly or approximately.* □

Here $\|T(Q)\|$ denotes the number of tuples in $T(Q)$.

Proof: (1) To see that $\mathsf{Chase}(Q,\mathcal{A})$ always terminates in $|Q| + 2\|Q\|$ steps, where $|Q|$ and $\|Q\|$ are the size and rows in $T(Q)$, respectively, observe the following. (i) Every step of a chasing sequence marks a variable or tuple in $T(Q)$. (ii) There are at most $|Q|$ variables and $\|Q\|$ tuples. (iii) Each variable and tuple can be marked only once and twice, respectively.

We next prove that $\mathsf{Chase}(Q,\mathcal{A})$ is Church-Rosser.

(a) We first show that for any two chasing sequences ℓ and ℓ' for Q with $\mathcal{A}$ (any access schema), the set of covered variables in $T(Q)$ is the same for ℓ and ℓ'. Suppose by contradiction that ℓ and ℓ' lead to different sets of 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 $\gamma_i = R(X \to Y, N)$ (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 covered variables and $x \in t[Y]$. Indeed, if this is not the case, then no variables in $S \setminus S'$ can be covered by applying chase rule (1). Since $x \notin S'$, we can append l_i with γ_i at the end of ℓ' to mark x as covered. This contradicts the assumption that ℓ' is a chasing sequence. Thus $S = S'$.

(b) We next show that the sets of exactly and approximately covered tuples in $T(Q)$ by ℓ and ℓ' are the same. This is obvious given that the set of constants and covered variables by ℓ and ℓ' are the same, guaranteed by (a).

(2) This immediately follows from the observation that access templates of type (a) given in the proof of Theorem 29(1-2) already make all tuples approximately covered.

□

Fetching plan from chase. From a chasing sequence for Q with $\mathcal{A}$, a fetching plan for Q under $\mathcal{A}$ can be derived. For each tuple t_S of relation S in $T(Q)$, let $T_i(Q) \overset{\gamma_i}{\mapsto} T_{i+1}(Q)$ be the step that marks t_S covered. Then a fetching plan ξ_S^F for S of 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 a plan that fetches variables in V (possibly with $\times$ and π_Y). The fetching plan ξ_Q^F for Q under $\mathcal{A}$ collects all such ξ_S^F for all relation names S in Q.

Example 17: From the chasing sequence of Fig. 4.3, a fetching plan $\xi_{Q_3}^F$ for Q_3 under $\mathcal{A}_3$ is derived as follows:

$T_1 = \mathsf{fetch}(A \in \{2\}, R, B, \psi_1)$;
$T_2 = \mathsf{fetch}(CE \in \{1\} \times \pi_B(T_1), S, F, \psi_2)$;
$T_3 = \mathsf{fetch}(A \in \pi_F(T_2), R, B, \psi_1)$;
$T_4 = \mathsf{fetch}(CEF \in T_2, S, G, \varphi_1)$.

Here $\xi_{R_1}^F = T_1$, $\xi_S^F = (T_1, T_2, T_4)$, $\xi_{R_2}^F = (T_1, T_2, T_3)$. □

Evaluation plan. The evaluation plan ξ_Q^E for Q under $\mathcal{A}$ conducts the same relational operations in Q except that it relaxes selection conditions in Q, as follows. (a) For each $\sigma_{A=c}$, if A is fetched via template $R(X \to Y, k, \bar{d}_Y(k))$ in $\mathcal{A}$, where $A \in Y$ and c is a constant, then replace it with $\sigma_{|\mathsf{dis}_A(A,c)| \leqslant \bar{d}_Y(k)[A]}$. (b) For each $\sigma_{A=B}$, if both A and B are fetched via templates with resolution tuples $\bar{d}_1(k_1)$ and $\bar{d}_2(k_2)$, then replace it with $\sigma_{|\mathsf{dis}(A,B)| \leqslant \bar{d}_1(k_1)[A] + \bar{d}_2(k_2)[B]}$. Similarly for the case when A or B is fetched via templates.

For example, given the fetching plan of Example 17, the evaluation plan for Q_3 under $\mathcal{A}_3$ consists of exactly the same operations in Q_3 since all attributes in selections are covered.

One can easily verify the following.

Lemma 33: *For any* SPC *query Q and access schema $\mathcal{A}$, if there is a chasing sequence that covers every tuple in $T(Q)$, then (ξ_Q^F, ξ_Q^E) is a query plan for Q under $\mathcal{A}$.* □

Proof: To show that (ξ_Q^F, ξ_Q^E) is a query plan for Q under $\mathcal{A}$, we only need to consider the case when we instantiate all parameters k in access templates to their maximum when using them in the plan, by the definition of query plans under access schema. Since for a template $R(X \to Y, k, \bar{d}_Y(k))$, when k is set to its maximum, it becomes an access constraint in essence. Hence we can treat all templates in $\mathcal{A}$ as constraints. In other words, it suffices to show the following:

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 (ξ_Q^F, ξ_Q^E) is a query plan for Q under $\mathcal{A}$,* i.e., *for any $D \models \mathcal{A}$, $\xi_Q(D) = Q(D)$, where $\xi_Q = (\xi_Q^F, \xi_Q^E)$.*

We prove this as follows. (1) We first translate chasing sequences into fetching plans, which, together with evaluation plans, constitute query plans under $\mathcal{A}$. (2) We then (equivalently) interpret the translated query plans ξ in terms of tableau queries called *representing query* Q_ξ of ξ. (3) We finally show that $Q_\xi \equiv Q$ when all tuples in $T(Q)$ are covered by the chasing sequence that derives ξ in step (1).

(1) Fetching plans from chasing sequences. We group *w.l.o.g.* deduction steps in the chasing sequence that share the same "inputs", *i.e.,* the set $t[X]$ of constants and covered variables and access constraint $\gamma = R(X \to Y, N)$ used for deduction via rule (1) of the chase. 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 deduction step. For a chasing sequence $\ell = s_1, \ldots, s_n$ like this, we construct a fetching plan ξ_ℓ^F by translating each step s_i into a fetch operation as follows.

- For each deduction s_i ($i \in [1, n]$) in ℓ, if s_i marks variables in t over R by chase rule (1) or tuple t by chase rule (2), via constraint $\psi = R(X \to Y, N)$, then s_i is translated to $\mathsf{fetch}(X \in \xi_{t[X]}, R, Y, \psi)$, where $\xi_{t[X]}$ is the combination of constants and fetching sub-plans for variables in $t[X]$, via projection and Cartesian-Product.

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

(2) Fetching plans ξ_ℓ as representing queries Q_ξ. We interpret $\xi_\ell = (\xi_\ell^F, \xi_Q^E)$ as a tableau query, referred to as the *representing query* Q_ξ of Q and defined as follows.

We construct Q_ξ in its tableau form from ξ_ℓ as follows.

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

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

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

Intuitively, (i) those tuples t' of $T(Q_\xi)$ processed in step (c) correspond to the evaluation plan ξ_Q^E; and (ii) all new tuples t' in step (b) are to encode the fetching plan ξ_ℓ.

Since all tuples t in $T(Q)$ are covered by the chasing sequence ℓ, after step (c), tuples in $T(Q)$ will be completely removed from $T(Q_\xi)$. Thus, by the semantics of fetch operation, $(T(Q_\xi), u(Q)) \equiv_{\mathcal{A}} \xi_\ell$ ($Q_\xi(D) = Q(D)$ for any $D \models \mathcal{A}$) when all tuples in $T(Q)$ are covered by ℓ, *i.e.,* $T(Q_\xi)$ encodes the derived query plan ξ_ℓ from the chasing sequence ℓ. Here $Q_1 \equiv_{\mathcal{A}} Q_2$ denotes $\mathcal{A}$-equivalence, *i.e.,* for all datasets D that satisfy $\mathcal{A}$, $Q_1(D) = Q_2(D)$; it is a notion stronger than the conventional query equivalence (see [AHV95]) by incorporating access schema $\mathcal{A}$ (Recall Chapter 2)

(3) Equivalence between Q_ξ and Q. We next show that $Q_\xi \equiv Q$ (via the conventional notion of query equivalence [AHV95]).

(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 in step (c) of (2) corresponding to t. Observe 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 [AHV95].

(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, $\rho'(t'[A]) = t[A]$ for each attribute A. Clearly ρ' is a homomorphism from $(T(Q_\xi), u(Q))$ to $(T(Q), u(Q))$ as $T(Q_\xi)$ 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 things together, we have that $\xi_\ell \equiv_{\mathcal{A}} Q_\xi \equiv Q$. □

From Lemmas 32 and 33 it follows that under the access schema $\mathcal{A}_0$ of Theorem 29, for any SPC query Q, there always exists a canonical query plan (ξ_Q^F, ξ_Q^E).

Algorithm parQP$_{\mathsf{SPC}}$. Putting these together, we present algorithm parQP$_{\mathsf{SPC}}$ in Fig. 4.4. Given the access schema $\mathcal{A}_0$ and an SPC query Q, it first computes a chasing sequence ℓ for Q with $\mathcal{A}_0$ via Chase($Q, \mathcal{A}$) (line 1). It optimizes ℓ by minimizing approximately covered attributes. For each tuple t of relation R in $T_i(Q)$ that is not exactly covered, it picks a template $R(X \to Y, k, \bar{d}(k))$ in $\mathcal{A}_0$ such that it covers t and $|Y|$ is minimum (line 2). It then derives fetching plan ξ_Q^F for Q from the optimized ℓ (line 3), and evaluation plan ξ_Q^E *w.r.t.* ξ_Q^F (line 4). It returns ξ_Q = (ξ_Q^F, ξ_Q^E) (line 5).

Example 18: for query Q_2 of Example 13 and $\mathcal{A}_2$ of Example 14, algorithm parQP$_{\mathsf{SPC}}$ generates fetching plan $\xi_{Q_2}^F$:

T_1 = fetch(pid $\in \{p_0\}$, friend, fid, ψ_1);

```
Algorithm parQP_SPC
Input: SPC query $Q$ and the access schema $\mathcal{A}_0$ of Theorem 29.
Output: A query plan $\xi_Q$ for $Q$ under $\mathcal{A}_0$.
1. $\ell$ := Chase$(Q, \mathcal{A}_0)$; /* $\ell$ is a chasing sequence for $Q$ */
2. optimize $\ell$ for relations in $Q$ that are approximately covered;
3. generate fetching plan $\xi_Q^F$ from the optimized $\ell$;
4. generate evaluation plan $\xi_Q^E$ for $Q$ w.r.t. $\xi_Q^F$;
5. return $\xi_Q = (\xi_Q^F, \xi_Q^E)$;
```

Figure 4.4: Algorithm $\mathsf{parQP}_{\mathsf{SPC}}$

$T_2 = \mathsf{fetch}(\mathsf{pid} \in \pi_{\mathsf{fid}}(T_1), \mathsf{person}, \mathsf{city}, \psi_2)$;

$T_3 = \mathsf{fetch}(\mathsf{city} \in \pi_{\mathsf{city}}(T_2), \mathsf{poi}, (\mathsf{id}, \mathsf{type}, \mathsf{street}), \varphi_1)$.

Here $\xi^F_{\mathsf{friend}} = T_1$, $\xi^F_{\mathsf{person}} = (T_1, T_2)$, $\xi^F_{\mathsf{poi}} = (T_1, T_2, T_3)$. The evaluation plan s Q_2 by relaxing condition type = "cinema" with $|\mathsf{dis}_{\mathsf{type}}(\mathsf{type}, \text{"cinema"})| \leqslant \bar{d}_{\mathsf{street}}(k)[\mathsf{type}]$.
□

Correctness & Complexity. The correctness of $\mathsf{parQP}_{\mathsf{SPC}}$ is warranted by Lemmas 32 and 33. The algorithm takes $O(|Q|\|\mathcal{A}\|)$ time, since both Chase$(Q, \mathcal{A})$ and the optimization can be implemented in $O(|Q|\|\mathcal{A}\|)$ time, by building an inverted index that takes $O(1)$ time to find an constraint or template of $\mathcal{A}$ to use in each chasing step, similar to the linear time algorithm for FD implication.

4.4.2 Generating α-bounded Execution Plans

We next present algorithm $\mathsf{instQP}_{\mathsf{SPC}}$. Given a query plan ξ_Q for an SPC Q found by $\mathsf{parQP}_{\mathsf{SPC}}$, a resource ratio $\alpha \in (0,1]$ and a database $D \models \mathcal{A}_0$, it computes an α-bounded execution plan ξ_* of ξ_Q in D, and an C bound η.

Execution plan generation is essentially an optimization problem, to maximize η when instantiating ξ_Q. Its decision problem, denoted by MAEP, is stated as follows.

- Input: ξ_Q, $\mathcal{A}_0$, α, D as above, and a bound $\eta \in [0,1]$.
- Question: Does there exist an α-bounded execution plan ξ_* of ξ_Q in D such that $\mathsf{F}_{\mathsf{rel}}(\xi_*(D), Q, D) \geqslant \eta$ and $\mathsf{F}_{\mathsf{cov}}(\xi_*(D), Q, D) \geqslant \eta$?

The problem is Σ_3^p-hard. Here Σ_3^p is the complexity class at the third level of the polynomial hierarchy beyond NP unless P = NP (see [Sto76]).

$$I_{01} = \begin{array}{|c|} X \\ \hline 1 \\ 0 \end{array} \quad I_{\vee} = \begin{array}{|ccc|} B & A_1 & A_2 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{array} \quad I_{\wedge} = \begin{array}{|ccc|} B & A_1 & A_2 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{array} \quad I_{\neg} = \begin{array}{|cc|} A & \bar{A} \\ \hline 0 & 1 \\ 1 & 0 \end{array} \quad I_s = \begin{array}{|c|} A \\ \hline 0 \\ 1 \end{array}$$

Figure 4.5: Relation instances used in the proof of Theorem 34.

Theorem 34: *Problem* MAEP *is Σ_3^p-hard.* □

Proof: We show that MAEP is Σ_3^p-hard by reduction from the $\exists^*\forall^*\exists^*$3CNF problem, which is known to be Σ_3^p-complete [Sto76]. The $\exists^*\forall^*\exists^*$3CNF 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.

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

(1) The database D consists of six relations specified by relation schemas $R_{01}(B)$, $R_V(B,A_1,A_2)$, $R_{\wedge}(B,A_1,A_2)$, $R_{\neg}(A,\bar{A})$, $R_s(A)$ and $R_X(I,X)$. Their instances correspond to the first five relations shown in Fig. 4.5. 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 type (a) access templates specified in the proof of Theorem 29(1-2); we use no access constraints in the proof.

(3) We define the SPC query Q as follows, expressed in 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 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) Let the query plan for Q under $\mathcal{A}$ be $\xi_Q = (\xi_Q^F, \xi_Q^E)$, where ξ_Q^F is a sequence of fetching operation $\mathsf{fetch}(\emptyset, R, \mathsf{attr}(R), \varphi_R, k)$ for each access template φ_R for each relation schema R. The evaluation plan ξ_Q^E is by definition Q itself. Note that ξ_Q is the output of $\mathsf{parQP}_{\mathsf{SPC}}(Q,\mathcal{A})$.

(5) We set $\alpha = \dfrac{13+m}{14+2m}$. 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).

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

We next verify that there exists an α-bounded execution plan ξ_* of ξ_Q and an index $I_\mathcal{A}$ for $\mathcal{A}$ carried by D such that $\mathsf{F}_{\mathsf{rel}}(\xi_*(D),Q,D) = \mathsf{F}_{\mathsf{cov}}(\xi_*(D),Q,D) = 1 = \eta$ with $I_\mathcal{A}$ if and only if ϕ is true.

$\boxed{\Rightarrow}$ 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 $Q(D) \neq \emptyset$. Thus, by the definition of C-measure, if $\mathsf{F}_{\mathsf{rel}}(\xi_*(D),Q,D) = \mathsf{F}_{\mathsf{cov}}(\xi_*(D),Q,D) = \eta = 1$, then $\xi_*(D) = Q(D)$. Let D_{ξ_*} be the output of the instantiated fetching plan ξ_Q^F in ξ_* in D. Then $\xi_*(D) = Q(D_{\xi_*})$. By the definition of Q, whenever D_{ξ_*} is obtained from D by removing tuples from I_{01}, $I_\wedge$, $I_\vee$ and $I_\neg$, then $\xi_*(D) = Q(D_{\xi_*}) \neq Q(D)$. Similarly, when D_{ξ_*} does not contain (1) in I_s', then $\xi_*(D) = Q(D_{\xi_*}) = \emptyset$. Hence, if ξ_* accesses only D_{ξ_*} of D and in addition, $Q(D) = \xi_*(D) = Q(D_{\xi_*})$, ξ_* 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 $\xi_*(D) = Q(D_{\xi_*}) = \emptyset$ if there exists an $i \in [1,m]$ such that neither $(i,0)$ nor $(i,1)$ is present in D_{ξ_*}. Thus, D_{ξ_*} must be of cardinality at least $13+m$. Since $\alpha = \dfrac{13+m}{14+2m}$ and $|D| = 14+2m$, this implies that D_{ξ_*} 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_*}; similarly for I'_X.

We claim that if $Q(D) = \xi_*(D) = Q(D_{\xi_*})$ 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, $Q(D) = Q(D_{\xi_*})$ implies that all truth assignments of Y are returned by $Q(D_{\xi_*})$, 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.

$\boxed{\Leftarrow}$ Suppose that φ is true. Let μ^0_X 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 $\varphi_S = S(\emptyset \to \mathsf{attr}(S), k, \bar{d}_{\mathsf{attr}(S)}(k))$ for relations I_{01}, $I_\vee$, $I_\wedge$, $I_\neg$ are constructed by arbitrarily sampling tuples for each k. For instance, when S is $R_\vee$, $\mathsf{fetch}(\emptyset, R_\vee, BA_1A_2, \varphi_{R_\vee}, 0)$ can simply return (0, 0, 0).
- The index for template $\varphi_{R_s} = R_s(\emptyset \to A, k, \bar{d}_A(k))$ is as follows: when k = 0, it returns 1; and when k = 1, it retrieves $\{0, 1\}$.
- The index for $\varphi_{R_X} = R_X(\emptyset \to IX, k, \bar{d}_{IX}(k))$ is as follows: for $k = \log_2^m$, it returns $(i, \mu^0_X(x_i))$ 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 execution plan ξ_* of ξ_Q in D for Q is constructed as follows:

- for parameters k in the fetch operations for relations other than R_s and R_X, set k to be their maximum;
- for $\mathsf{fetch}(\emptyset \to A, R_s, \varphi_{R_s}, k)$, set k = 0;
- for $\mathsf{fetch}(\emptyset \to IX, R_X, \varphi_{R_X}, k)$, set $k = \log_2 m$.

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

In light of the intractability, we develop a PTIME heuristic as $\mathsf{instQP}_{\mathsf{SPC}}$. It generates ξ_* by accessing *neither D nor* the indices of $\mathcal{A}_0$. Algorithm $\mathsf{instQP}_{\mathsf{SPC}}$ utilizes a lower bound function L for ξ_Q to control the parameters k's of the templates of $\mathcal{A}_0$ used in ξ_Q, such that the accuracy of $\xi_*(D)$ is above L. Guided by L, $\mathsf{instQP}_{\mathsf{SPC}}$ greedily instantiates k's and increases L, until it finds ξ_* with data tariff $\alpha|D|$.

More specifically, function L is given as $\frac{1}{1+\mathsf{max}(d_{\mathsf{rel}},d_{\mathsf{cov}})}$. Here d_{rel} and d_{cov} are upper bounds on the relevance and coverage distances of approximate answers. The upper bounds are inductively defined as follows, based on the structure of query Q *w.r.t.* the fetching plan ξ_Q^F of ξ_Q.

(1) Q is R. Then $d_{\mathsf{rel}}(Q) = 0$; $d_{\mathsf{cov}}(Q) = \Sigma_{A\in R} d_{\mathsf{cov}}(Q)[A]$, where (a) $d_{\mathsf{cov}}(Q)[A] = \bar{d}_\delta(k_\delta)[A]$ if A is retrieved by fetch using template $R(X \to Y, k_\delta, \bar{d}_\delta(k_\delta))$ of $\mathcal{A}_0$ ($A \in Y$), or (b) $d_{\mathsf{cov}}(Q)[A] = 0$ if fetch uses a constraint in $\mathcal{A}_0$.

Intuitively, when Q is R, every approximate answer returned by an execution plan of ξ_Q is relevant to Q, *i.e.,* $d_{\mathsf{rel}}(Q) = 0$. The maximum distance between approximate answers and exact answers on attribute A does not exceed $\bar{d}_\delta(k_\delta)[A]$ if A is fetched using a template with parameter k_δ; and it is 0 if fetch uses an access constraint, *i.e.,* exact.

(2) Q is $\sigma_{S[A]=c}(Q')$. Then $d_{\mathsf{cov}}(Q) = d_{\mathsf{cov}}(Q')$, and $d_{\mathsf{rel}}(Q) = d_{\mathsf{rel}}(Q') + \bar{d}_\delta(k_\delta)[A]$ if $S[A]$ is fetched using access template $R(X \to Y, k_\delta, \bar{d}_\delta(k_\delta))$, and $d_{\mathsf{rel}}(Q) = d_{\mathsf{rel}}(Q')$ otherwise.

Intuitively, when Q is $\sigma_{S[A]=c}(Q')$, the worst coverage distance of Q is the same as that of Q'. The worst relevance distance between approximate answer s and Q is the distance between s and Q', plus resolution $\bar{d}_\delta(k_\delta)[A]$ if s is approximately covered and fetched using a template with parameter k_δ. If s is exactly covered, no extra distance is introduced by fetch.

(3) Q is $\pi_Y(Q')$.

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

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' relevant to attributes Y, since R_Q consists of Y only.

(4) 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 to Q_1 and for Q_2.; similarly for coverage distance.

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

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

```
Algorithm instQP_SPC
Input: SPC query $Q$, the access schema $\mathcal{A}_0$ of Theorem 29,
        query plan $\xi_Q$, resource ratio $\alpha$ and database $D$;
Output: An $\alpha$-bounded execution plan $\xi_*$ and accuracy bound $\eta$.
1. for each parameter $k_\delta$ in $\xi_Q$ do $k_\delta := 0$;
2. let $\xi_\alpha$ be the instantiation of $\xi_Q$ with $k_\delta$ for all fetch $\delta$;
3. $\xi_* := \xi_\alpha$; set $\eta$ to be the value of $L$ with $k_\delta$ for all $\delta$ in $\xi_Q$;
4. while $\mathsf{tariff}(\xi_\alpha) \leqslant \alpha|D|$ do
5.     $\xi_* := \xi_\alpha$; set $\eta$ to be the $L$ value w.r.t. $k_\delta$ for all fetch $\delta$;
6.     find $k_\delta$ such that $L(k_\delta := k_\delta + 1) \geqslant L(k_{\delta'} := k_{\delta'} + 1)$;
       /* for all $\delta'$; $L(k_\delta := k_\delta + 1)$ is the L value by adding 1 to $k_\delta$ */
7.     $k_\delta := k_\delta + 1$; update $\xi_\alpha$ with updated parameter $k_\delta$;
8. return $(\xi_*, \eta)$;
```

Figure 4.6: Algorithm instQP$_{\mathsf{SPC}}$

- $d_{\mathsf{cov}}(Q) = d_{\mathsf{cov}}(Q')$.

Here $h_{R[A]}$ = 1 (resp $h_{R'[B]}$ = 1) if $R[A]$ (resp. $R'[B]$) is not covered and R (resp. R') is approximately covered in ξ_Q, and $h_{R[A]}$ = 0 (resp. $h_{R'[B]}$ = 0) otherwise; δ and δ' are the fetch operations that retrieves R and R' data, respectively.

Observe that the lower bound distinguishes the following cases: (i) $R[A]$ is fetched via an access constraint (*i.e.,* $h_{R[A]} = 0$) and $R'[B]$ is fetched via an access template (*i.e.,* $h_{R'[B]} = 1$); (ii) $R[A]$ is fetched via template (*i.e.,* $h_{R[A]} = 1$) and $R'[B]$ is fetched via constraint (*i.e.,* $h_{R'[B]} = 0$); (iii) both $R[A]$ and $R'[B]$ are fetched via access constraints (*i.e.,* $h_{R[A]} = h_{R'[B]} = 0$); and (iv) both $R[A]$ and $R'[B]$ are fetched via access templates (*i.e.,* $h_{R[A]} = h_{R'[B]} = 1$). For case (i), $d_{\mathsf{rel}}(Q)$ is the sum of $d_{\mathsf{rel}}(Q')$ and the maximum error that can be introduced by the use of template on $R'[B]$, *i.e.,* $\bar{d}_{\delta'}(k_{\delta'})[B]$. Similar for the other three cases.

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.

Algorithm instQP$_{\mathsf{SPC}}$. Leveraging L, we present the algorithm in Fig. 4.6. It first initializes parameters $k_\delta = 0$ for all fetch operations δ (line 1). It then instantiates ξ_Q with these k_δ values, to get an execution plan ξ_α (line 2). It initializes $\xi_* = \xi_\alpha$, and sets η to be L value with these k_δ (line 3).

It then iteratively increases parameters k_δ while keeping ξ_α still α-bounded (lines 4-7). In each iteration, it checks whether the data tariff of ξ_α, computed by function $\mathsf{tariff}(\xi_\alpha)$, does not exceed $\alpha|D|$ (line 4). If so, it upgrades ξ_* to ξ_α, and updates η to take the current L value (line 5). It then identifies a parameter k_δ such that increasing it (by 1) will yield the maximum L value among all parameters (line 6). It increases the identified parameter k_δ by 1 and updates ξ_α by instantiating ξ_Q with the updated k_δ (line 7). It returns ξ_* and η when $\mathsf{tariff}(\xi_\alpha)$ reaches $\alpha|D|$ (line 8). The accuracy η has a provable lower bound.

Theorem 35: *For ξ_* and η generated by* $\mathsf{instQP_{SPC}}$,

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

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

Here (1) for each access template $\varphi = R(X \to Y,k,\bar{d}_\varphi(k))$ in $\mathcal{A}_0$, we denote $\mathsf{max}_{A\in Y}\bar{d}_\varphi(k)[A]$ as $\bar{d}^m_\varphi(k)$; (2) for a query Q, $\|Q\|$ denotes the number of relations in 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.

Proof: The proof consists of two parts.

(1) We first show that $\mathsf{F_{rel}}(\xi_*(D),Q,D) \geqslant \eta$ and $\mathsf{F_{cov}}(\xi_*(D),Q,D) \geqslant \eta$. It suffices to show that for any instantiation $\bar{k}$ to parameters in ξ_Q, $\mathsf{F_{rel}}(\xi_Q(\bar{k})(D),Q,D) \geqslant \frac{1}{1+d_{\mathsf{rel}}(Q)}$ and $\mathsf{F_{cov}}(\xi_Q(\bar{k})(D),Q,D) \geqslant \frac{1}{1+d_{\mathsf{cov}}(Q)}$. Here $\xi_Q(\bar{k})$ denotes the execution plan of ξ_Q with parameters instantiated to $\bar{k}$. We show this by induction on the structure of Q.

<u>*Basis*</u>. When $T(Q_\xi)$ contains of only one tuple t, by the definition of relevance and coverage ratio, and lower bound function L, for any k, we have that $\mathsf{F_{rel}}(\xi_Q(\bar{k})(D),Q,D) \geqslant \frac{1}{1+d_{\mathsf{rel}}(\bar{k})}$ and $\mathsf{F_{cov}}(\xi_Q(\bar{k})(D),Q,D) \geqslant \frac{1}{1+d_{\mathsf{cov}}(\bar{k})}$.

<u>*Induction steps*</u>. We next consider more complex Q.

(a) When $Q = \sigma_{R[A]=c}(Q')$. There are two cases. (i) If $R[A]$ is fetched by access constraints, *i.e.,* exactly covered, then by the definition of C-measure, we know $\mathsf{F_{rel}}(\xi_Q(\bar{k})(D),Q,D) = \mathsf{F_{rel}}(\xi_{Q'}(\bar{k})(D),Q',D) \geqslant d_{\mathsf{rel}}(Q') = d_{\mathsf{rel}}(Q)$ and $\mathsf{F_{cov}}(\xi_Q(\bar{k})(D),Q,D) = \mathsf{F_{cov}}(\xi_{Q'}(\bar{k})(D),Q,D) \geqslant \frac{1}{1+d_{\mathsf{cov}}(Q')} = \frac{1}{1+d_{\mathsf{cov}}(Q)}$. (ii) If $R[A]$ is fetched via an access template instantiated with k_A, let $\bar{k} = \bar{k}' \cup \{k_A\}$, where $\bar{k}'$ is the instantiation of the sub-plan for Q'. Then

$$\begin{aligned}
&\mathsf{F}_{\mathsf{rel}}(\xi_Q(\bar{k})(D),Q,D) \geqslant \\
&\quad \frac{1}{1+(\bar{d_A}(k_A)[A]+\frac{1}{\mathsf{F}_{\mathsf{rel}}(\xi_{Q'}(\bar{k}')(D),Q',D)}-1)} \geqslant \\
&\quad \frac{1}{\bar{d_A}(k_A)[A]+d_{\mathsf{rel}}(Q')+1} = \\
&\quad \frac{1}{1+d_{\mathsf{rel}}(Q)}.
\end{aligned}$$

Since $R_Q = R_{Q'}$, by the definition of coverage ratio, we have that $\mathsf{F}_{\mathsf{cov}}(\xi_Q(\bar{k})(D),Q,D) = \mathsf{F}_{\mathsf{cov}}(\xi_{Q'}(\bar{k}')(D),Q',D) \geqslant \frac{1}{1+d_{\mathsf{cov}}(Q')} = \frac{1}{1+d_{\mathsf{cov}}(Q)}$. Note that this holds since the fetch operation that fetches $R[A]$ is guaranteed to return a non-empty set of tuples, ensured by the condition of chasing rule (c) in Section 4.4.1.

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

(b) When $Q = \sigma_{S[A]=S'[B}(Q')$. There are four cases, corresponding to the definition of the lower bound function L for $Q = \sigma_{S[A]=S'[B]}(Q')$. We consider the case when both $S[A]$ and $S'[B]$ are fetched via templates instantiated with k_A and k_B, respectively. Let $\bar{k} = \bar{k}' \cup \{k_A, k_B\}$, where $\bar{k}'$ is the instantiation of the sub-plan for Q'. Then by the definition of relevance ratio, we have the following:

$$\begin{aligned}
&\mathsf{F}_{\mathsf{rel}}(\xi_Q(\bar{k})(D),Q,D) \geqslant \\
&\quad \frac{1}{1+(\bar{d_A}(k_A)[A]+\bar{d_B}(k_B)[B]+\frac{1}{\mathsf{F}_{\mathsf{rel}}(\xi_{Q'}(\bar{k}')(D),Q',D)}-1)} \geqslant \\
&\quad \frac{1}{\bar{d_A}(k_A)[A]+\bar{d_B}(k_B[B])+d_{\mathsf{rel}}(Q')+1} = \\
&\quad \frac{1}{1+d_{\mathsf{rel}}(Q)}.
\end{aligned}$$

Similar to (a), $\mathsf{F}_{\mathsf{cov}}(\xi_Q(\bar{k})(D),Q,D) = \mathsf{F}_{\mathsf{cov}}(\xi_{Q'}(\bar{k}')(D), Q',D) \geqslant \frac{1}{1+d_{\mathsf{cov}}(Q')} = \frac{1}{1+d_{\mathsf{cov}}(Q)}$.

(c) When $Q = Q_1 \times Q_2$. Let $\bar{k} = (\bar{k}_1, \bar{k}_2)$ be any instantiation to parameters of fetch operations in ξ_Q, where $\bar{k}_1$ and $\bar{k}_2$ are the instantiation to sub-plans of ξ_Q for Q_1 and Q_2, respectively. Then

$$\begin{aligned}
&\mathsf{F}_{\mathsf{rel}}(\xi_Q(\bar{k})(D),Q,D) = \\
&\quad \frac{1}{1+(\frac{1}{\mathsf{F}_{\mathsf{rel}}(\xi_{Q_1}(\bar{k}_1)(D),Q_1,D)}-1)+(\frac{1}{\mathsf{F}_{\mathsf{rel}}(\xi_{Q_2}(\bar{k}_2)(D),Q_2,D)}-1)} \geqslant \\
&\quad \frac{1}{1+d_{\mathsf{rel}}(Q_1)+d_{\mathsf{rel}}(Q_2)} = \\
&\quad \frac{1}{1+d_{\mathsf{rel}}(Q)}.
\end{aligned}$$

Similarly, by the definition of coverage ratio, we have

$$\mathsf{F}_{\mathsf{cov}}(\xi_Q(\bar{k})(D), Q, D) = \frac{1}{1 + (\frac{1}{\mathsf{F}_{\mathsf{cov}}(\xi_{Q_1}(\bar{k}_1)(D), Q_1, D)} - 1) + (\frac{1}{\mathsf{F}_{\mathsf{cov}}(\xi_{Q_2}(\bar{k}_2)(D), Q_2, D)} - 1)} \geqslant \frac{1}{1 + d_{\mathsf{cov}}(Q_1) + d_{\mathsf{cov}}(Q_2)} = \frac{1}{1 + d_{\mathsf{cov}}(Q)}.$$

(d) When $Q = \pi_Y(Q')$. Let $\bar{k}$ be any instantiation to parameters in ξ_Q, and $\bar{k}' = \pi_Y(\bar{k})$. By the definition of relevance ratio, $\mathsf{F}_{\mathsf{rel}}(\xi_Q(\bar{k})(D), Q, D) \geqslant \mathsf{F}_{\mathsf{rel}}(\xi_{Q'}(\bar{k})(D), Q, D) \geqslant \frac{1}{1+d_{\mathsf{rel}}(Q')} = \frac{1}{1+d_{\mathsf{rel}}(Q)}$. By the definition of coverage ratio, $\mathsf{F}_{\mathsf{cov}}(\xi_Q(\bar{k})(D), Q, D) \geqslant \mathsf{F}_{\mathsf{cov}}(\xi_{Q'}(\bar{k})(D), Q, D) \geqslant \frac{1}{1+d_{\mathsf{cov}}(Q)}$, since every attribute A in R_Q is fetched by a fetching sub-plan of ξ_Q containing only one fetch with access template.

Therefore, $\mathsf{F}_{\mathsf{rel}}(\xi_*(D), Q, D) \geqslant \frac{1}{1+d_{\mathsf{rel}}(Q)} \geqslant \eta$ and $\mathsf{F}_{\mathsf{cov}}(\xi_*(D), Q, D) \geqslant \frac{1}{1+d_{\mathsf{cov}}(Q)} \geqslant \eta$, since $\eta = \frac{1}{1+\mathsf{max}(d_{\mathsf{rel}}(Q), d_{\mathsf{cov}}(Q))}$.

(2) We next show the lower bound for η. Observe the following. (a) At least there exists one fetch operation in ξ_Q whose parameter is instantiated to $k^* + 1$. Because of this, (b) by the greedily guided instantiation of parQP$_{\mathsf{SPC}}$, the relevance distance introduced by every fetch operation in ξ_* is no larger than $d_0 = \mathsf{max}_{\varphi \in \mathcal{A}_0} \bar{d}^m_{\varphi}(k^*)$. Therefore, (c) by the definition of 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, $\eta \geqslant \frac{1}{1+\mathsf{max}(d^*_{\mathsf{rel}}, d^*_{\mathsf{cov}})}$. □

Correctness & Complexity. The correctness of instQP$_{\mathsf{SPC}}$ is guaranteed by Theorem 35. The algorithm is in $O(|Q|\|\mathcal{A}\| + |Q|\|Q\|\lceil\log_2 \alpha|D|\rceil)$ time. In particular, the **while** loop runs at most $\lceil\log_2 \alpha|D|\rceil$ times, and each time it takes $O(|Q|\|Q\|)$ time, where $|Q|$ is the size of Q.

4.5 Approximating RA Queries

To prove Theorem 29(3b), we develop a resource-bounded approximation scheme for RA under the access schema $\mathcal{A}_0$ of Theorem 29(1-2). The tricky part for RA is to enforce the semantics of set difference $Q_1 - Q_2$: its evaluation plan should ensure that for any approximate answer s returned for $Q_1 - Q_2$ in any database D, $s \notin Q_2(D)$, *without accessing the entire D*.

The scheme consists of two algorithms: (1) parQP$_{\mathsf{RA}}$ that extends parQP$_{\mathsf{SPC}}$ to generate canonical query plans ξ_Q for RA queries under $\mathcal{A}_0$, with a more involved evaluation plan due to set difference (Section 4.5.1); and (2) $\mathsf{instQP}_{\mathsf{RA}}$ that extends $\mathsf{instQP}_{\mathsf{SPC}}$ to derive an α-bounded execution plan from ξ_Q and compute an accuracy bound for RA (Section 4.5.2).

4.5.1 Generating Canonical Query Plans for RA

Canonical plans of Lemma 31 remain a normal form for RA query plans. Hence it suffices for algorithm $\mathsf{parQP}_{\mathsf{RA}}$ to generate canonical plan $\xi_Q = (\xi_Q^F, \xi_Q^E)$ for an RA query Q.

Lemma 36: *Under the access schema $\mathcal{A}_0$ of Theorem 29, every* RA *query Q has a canonical bounded query plan.* □

Proof: 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 31. Indeed, 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; (b) the evaluation plan is the group of evaluation plans for all max SPC sub-queries of Q, followed by a revision of Q where each of its max SPC sub-query Q_s is 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. □

Fetching plan. Algorithm $\mathsf{parQP}_{\mathsf{RA}}$ generates fetching plan ξ_Q^F for Q based on the following notion. A *max* SPC *sub-query* of Q is a sub-query Q_s of Q such that (a) Q_s is an SPC query, and (b) there exists no sub-query Q_s' of Q such that it is also in SPC, $Q_s \neq Q_s'$ and Q_s is a sub-query of Q_s'. Obviously, a fetching plan for all Q_s's of Q suffices to retrieve all the data needed to compute $Q(D)$. Hence $\mathsf{parQP}_{\mathsf{RA}}$ first generates fetching plans for all max SPC sub-queries of Q via $\mathsf{parQP}_{\mathsf{SPC}}$, and groups these plans together to make ξ_Q^F.

Evaluation plan. Given ξ_Q^F, $\mathsf{parQP}_{\mathsf{RA}}$ derives an evaluation plan ξ_Q^E for Q under $\mathcal{A}$. It "implements" an RA query $\mathsf{E}(Q)$, defined inductively based on the structure of Q as follows.

(1) Q is R: Then $\mathsf{E}(Q) = Q$.

(2) Q is $\sigma_{S[A]=c}(Q')$. Then $\mathsf{E}(Q) = \sigma_C(\mathsf{E}(Q'))$, where C is $|\mathsf{dis}_A(S[A], c)| \leqslant \bar{d}_Y(k)[A]$ if $S[A]$ is fetched via template $S(X \to Y, k, \bar{d}_Y(k))$ in $\mathcal{A}_0$; and it is $S[A] = c$ otherwise.

(3) Q is $Q_1 - Q_2$. Then $\mathsf{E}(Q) = \mathsf{E}(Q_1) - \pi_{R_{Q_1}} \sigma_C(\mathsf{E}(Q_1) \times \mathsf{E}(\widehat{Q}_2))$, where $\widehat{Q}_2$ is a query

that "expands" Q_2 to enforce the semantics of set difference together with condition C (to be given shortly). To give $\widehat{Q}_2$, we define the following.

(a) A *positive induced query* Q' of Q is a query such that

(i) the query tree (cf. [AHV95]) $T_{Q'}$ of Q' 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* in $T_{Q'}$.

Note that Q' is not necessarily a sub-query of Q. For example, $Q_1 \times Q_3$ is a positive induced query of $(Q_1 - Q_2) \times (Q_3 - Q_4)$, but it is not a sub-query of the latter.

(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 next define the *selection condition* C in $\mathsf{E}(Q)$. Let R_Q be the output schema of Q (the same as R_{Q_1} and R_{Q_2} for Q_1 and Q_2, respectively). Define $d(A) = 0$ if $R_{Q_2}[A]$ is fetched via an access constraint in $\mathcal{A}$ for $\widehat{Q_2}$; and $d(A) = \bar{d}_\varphi(k_\varphi)[A]$ if it is via a template $R(X \to Y, k_\varphi, \bar{d}_\varphi)$ with $A \in Y$. Then

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

This enforces the set difference semantics and warrants a non-zero relevance accuracy.

Lemma 37: *For any access schema $\mathcal{A}$,* RA *$Q = Q_1 - Q_2$ and $D \models \mathcal{A}$, if Q has fetching plan ξ_Q^F and $t \in Q_2(D)$, then for any execution plan ξ of (ξ_Q^F, $\mathsf{E}(Q)$) in D, $t \notin \xi(D)$.* □

Proof: By the definition of coverage distance, we know that for any instantiation $\bar{k}$ to ξ_Q^F, for any $t \in Q_2(D)$, since $t \in \widehat{Q_2}(D)$, there must exist $s \in \mathsf{E}(\widehat{Q_2})(D_{\xi_Q^F})$ such that $|\mathsf{dis}_A(t[A], s[A])| \leqslant d(A)$, where $D_{\xi_Q^F}$ is the fetched fraction of data from D by ξ_Q^F instantiated with $\bar{k}$, *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 $D_{\xi_Q^F}$ 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 instantiation $\bar{k}$ to $(\xi_Q^F, \mathsf{E}(Q_1 - Q_2))$, let ξ_* be the instantiated execution plan *w.r.t.* $\bar{k}$, then $\xi_*(D) = \mathsf{E}(Q_1 - Q_2)(\xi_Q^F(D))$ contains no answers that possibly "cover" exact answers in $\widehat{Q_2}(D)$ (and thus those exact answers in $Q_2(D)$). In other words, for any $t \in Q_2(D)$, $t \notin \xi_*(D)$. □

Example 19: Consider RA query $Q_4 = \pi_B(R(1,B) - S(1,B))$, where R is fetched by $\mathsf{fetch}(A, R, B, \varphi(k_R))$ with $R(A \to B, k, \bar{d}(k_R))$; similar for S. Then $\mathsf{E}(Q_4) = \pi_B(R(1,B) - \pi_{R[AB]}\sigma_C(R(1,B) \times S(1,B)))$, where $C = |\mathsf{dis}_B(R[B], S[B])| \leqslant \bar{d}(k_R)[B] + \bar{d}(k_S)[B]$. This excludes tuples fetched for $R(1,B)$ that are in $S(1,B)$, as they are certainly within distance $\bar{d}(k_R)[B] + \bar{d}(k_S)[B]$ of some tuples fetched for S. □

Evaluation plans for $\sigma_{S[A]\leqslant c}(Q')$, $\sigma_{S[A]=S'[B]}(Q')$, $\sigma_{S[A]=S'[B]}(Q')$, $\pi_Y(Q')$ and $Q_1 \times Q_2$ are the same as their counterparts for SPC (see Section 4.4.1). For the case $Q = Q_1 \cup Q_2$, $E(Q) = E(Q_1) \cup E(Q_2)$.

Algorithm $\mathsf{parQP}_{\mathsf{RA}}$. Given an RA query Q and the access schema $\mathcal{A}_0$ of Theorem 29, $\mathsf{parQP}_{\mathsf{RA}}$ works as follows (not shown). It first generates fetching plans $\xi^F_{Q_s}$ for all max SPC sub-queries Q_s of Q, and derives a fetching plan ξ^F_Q for Q from these. It then generates the evaluation plan ξ^E_Q as above. It returns canonical query plan (ξ^F_Q, ξ^E_Q) for Q under $\mathcal{A}_0$.

Correctness & Complexity. One can verify that $\mathsf{parQP}_{\mathsf{RA}}$ is correct and takes $O(|Q|(|Q| + \|\mathcal{A}_0\|))$ time, similar to $\mathsf{parQP}_{\mathsf{SPC}}$; in particular, $\mathsf{E}(Q) \equiv Q$; here $\equiv$ denotes query equivalence.

4.5.2 Generating α-bounded Execution Plans for RA

We next present algorithm $\mathsf{instQP}_{\mathsf{RA}}$. It extends $\mathsf{instQP}_{\mathsf{SPC}}$ in the following. (1) It s the lower bound function L for set union and difference in Q. More specifically, when $Q = Q_1 - Q_2$, $d_{\mathsf{rel}}(Q) = d_{\mathsf{rel}}(Q_1)$ and $d_{\mathsf{cov}}(Q) = d_{\mathsf{cov}}(Q_1)$; similarly for $Q_1 \cup Q_2$. (2) It estimates the accuracy bound via the maximal induced query of Q, and rectifies it by incorporating the impact of set difference.

Algorithm $\mathsf{instQP}_{\mathsf{RA}}$. The algorithm is shown in Fig. 4.7. It works as follows. It first instantiates ξ_Q with L in exactly the same way as $\mathsf{instQP}_{\mathsf{SPC}}$ does, and gets an α-bounded execution plan ξ_*, with distance upper bound values d^*_{rel} and d^*_{cov} of L (line 1).

It then improves the accuracy bound for $\xi_*(D)$ as follows. It computes the maximal induced query $\widehat{Q}$ of Q (line 2), and generates a query plan $\xi_{\widehat{Q}}$ for $\widehat{Q}$ via $\mathsf{parQP}_{\mathsf{SPC}}$ (line 3). After these, it instantiates $\xi_{\widehat{Q}}$ with the same parameters as used in ξ_*, to get an execution plan $\widehat{\xi}_*$ and a coverage distance bound $\widehat{d_{\mathsf{cov}}}^*$ for $\widehat{Q}$ (line 4). Let S and S' be the answers to ξ_* and $\widehat{\xi}_*$ in D, respectively (line 5). It computes the "coverage distance" d' between tuples in S and S' for $\widehat{Q}$ (line 6). Treating $d' + \widehat{d_{\mathsf{cov}}}^*$ as the rectified upper bound on the coverage distance for S, it gets an accuracy bound for S together with d^*_{rel} (line 7). It finally returns (ξ_*, η) (line 8).

Example 20: Recall Q_4 from Example 19. Assume that approximate answers to R and S are $\{t_1 = (1{,}1), t_2 = (1{,}100)\}$ and $\{t_3 = (1{,}99)\}$, retrieved by $\mathsf{fetch}(A, R, B, \varphi_R(k_R))$ and $\mathsf{fetch}(A, S, B, \varphi_S(k_S))$, respectively. By the definition of L (line 1, $\mathsf{instQP}_{\mathsf{RA}}$), d^*_{cov} is $\bar{d}_R(k_R)[B]$ such that for any tuple t' in instance of R with $t'[A] = 1$, $\mathsf{min}(|\mathsf{dis}_B(1,$

```
Algorithm instQP_RA
Input: RA query $Q$, the access schema $\mathcal{A}_0$ of Theorem 29,
        query plan $\xi_Q$, resource ratio $\alpha$ and database $D$;
Output: An $\alpha$-bounded execution plan $\xi_*$ and accuracy bound $\eta$.
1. Instantiate $\xi_Q$ with $L$ in the same way as instQP_SPC does;
/*Let $\xi_*$ be the $\alpha$-bounded plan; $d^*_{\mathsf{rel}}$ and $d^*_{\mathsf{cov}}$ be the upper bounds of $L$*/
2. compute maximal induced $\widehat{Q}$ of $Q$, an SPC query;
3. $\xi_{\widehat{Q}}$ := parQP_SPC$(\widehat{Q}, \mathcal{A}_0)$;
4. let $\widehat{\xi}_*$ and $\widehat{d_{\mathsf{cov}}}^*$ be the instantiated plan of $\xi_{\widehat{Q}}$ and the coverage
   distance bound with the parameters in $\xi_*$, respectively;
5. $S := \xi_*(D)$; $S' := \widehat{\xi}_*(D)$; /* executing the $\alpha$-bounded plan $\xi_*$ */
6. $d' := \mathsf{max}_{t \in S'} \mathsf{min}_{s \in S} \delta_{\mathsf{cov}}(\widehat{Q}, t, s)$;
7. $\eta := \dfrac{1}{1 + \mathsf{max}(d^*_{\mathsf{rel}}, d' + \widehat{d_{\mathsf{cov}}}^*)}$;
8. return $(\xi_*, \eta)$;
```

Figure 4.7: Algorithm instQP$_{\mathsf{RA}}$

$t'[B])|, |\mathsf{dis}_B(100, t'[B])|) \leqslant \bar{d}_R(k_R)[B]$. 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^*_{cov} is not an upper bound on the coverage distance $\delta_{\mathsf{cov}}(Q_4, t, s)$.

To rectify this, instQP$_{\mathsf{RA}}$ first computes upper bound on coverage distance for answers to $\widehat{Q_4}$, which is d^*_{cov} above. It then computes the "coverage distance" for using t_1 alone to cover $\{t_1, t_2\}$, which is d' (line 6, instQP$_{\mathsf{RA}}$). It then treats $d^*_{\mathsf{cov}} + d'$ as an upper bound for $\delta_{\mathsf{cov}}(Q_4, t, s)$, which yields a lower bound on the coverage of the final answers. □

The accuracy of $\xi_*(D)$ has provable lower bounds.

Theorem 38: *For ξ_* and η generated by* instQP$_{\mathsf{RA}}$,

- $\mathsf{F}_{\mathsf{rel}}(\xi_*(D), Q, D) \geqslant \eta$ *and* $\mathsf{F}_{\mathsf{cov}}(\xi_*(D), Q, D) \geqslant \eta$*;*
- $\mathsf{F}_{\mathsf{rel}}(\xi_*(D), Q, D) \geqslant \frac{1}{1+(\mu_C(\widehat{Q})+2\mu_v(\widehat{Q}))\cdot \mathsf{max}_{\varphi \in \mathcal{A}_0} \bar{d}^m_{\varphi}(k^*)}$, *in which* $k^* = \lfloor \log_2 \frac{\alpha|D|}{\|Q\|} \rfloor - 1$*; and*
- $\eta > 0$ *when* $\xi_*(D) \neq \emptyset$,

where μ_C, mu_v, $\bar{d}^m_{\varphi}(k^*)$ are the same as in Theorem 35 □

Proof: (1) By Lemma 37 and the definition of relevance ratio, when $Q = Q_1 - Q_2$, for any instantiation $\bar{k}$ to parameters in ξ_Q, $\mathsf{F}_{\mathsf{rel}}(\xi_Q(\bar{k})(D), Q, D) = \mathsf{F}_{\mathsf{rel}}(\xi_{Q_1}(\bar{k}_1)(D), Q_1, D)$, where $\bar{k}_1$ is the part of $\bar{k}$ for fetch operations in the fetching sub-plan of ξ_Q for Q_1. Hence, along the same lines as the proof of Theorem 35, one can verify that $\mathsf{F}_{\mathsf{rel}}(\xi_*(D), Q, D) \geqslant \frac{1}{1+d^*_{\mathsf{rel}}} \geqslant \eta$, where d^*_{rel} is the value of d_{rel} at line 1 of $\mathsf{instQP}_{\mathsf{RA}}$, and η is the lower bound returned by $\mathsf{instQP}_{\mathsf{RA}}$ in the end.

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

Lemma 39: *For any $D \models \mathcal{A}$ and* RA *query Q, consider ξ_*, $\widehat{\xi_*}$, $\widehat{d_{\mathsf{cov}}}^*$ and d' in algorithm* $\mathsf{instQP}_{\mathsf{RA}}$ *for Q and $\widehat{Q}$,*

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

(b) $\mathsf{F}_{\mathsf{cov}}(\widehat{\xi_*}(D), \widehat{Q}, D) \geqslant \frac{1}{1+\widehat{d_{\mathsf{cov}}}^*}$.

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

□

Indeed, (a) is ensured by the definition of maximal induced queries; (b) follows from the proof of Theorem 35(1) (observe that $\widehat{Q}$ is an SPC); and (c) follows from (a) and the definition of coverage ratio in the C-measure.

By (b), for any $t \in \widehat{Q}(D)$, there exists tuple $\widehat{s} \in \widehat{\xi_*}(D)$ such that $\delta_{\mathsf{cov}}(\widehat{Q}, t, \widehat{s}) \leqslant \widehat{d_{\mathsf{cov}}}^*$. By line 6 of algorithm $\mathsf{instQP}_{\mathsf{RA}}$, for any $\widehat{s} \in \widehat{\xi_*}(D)$, there exists $s \in \xi_*(D)$ such that $\delta_{\mathsf{cov}}(\widehat{Q}, \widehat{s}, s) \leqslant d'$. Therefore, for any $t \in \widehat{Q}(D)$, there exists $s \in \xi_*(D)$ such that $\delta_{\mathsf{cov}}(\widehat{Q}, t, s) \leqslant \widehat{d_{\mathsf{cov}}}^* + d'$. By the definition of coverage ratio, we have that $\mathsf{F}_{\mathsf{cov}}(\xi_*(D), \widehat{Q}, D) \geqslant \frac{1}{1+\widehat{d_{\mathsf{cov}}}^*+d'} = \eta$. By (c), $\mathsf{F}_{\mathsf{cov}}(\xi_*(D), Q, D) \geqslant \eta$.

(2) Observe that $\mathsf{F}_{\mathsf{rel}}(\xi_*(D), Q, D) \geqslant \mathsf{F}_{\mathsf{rel}}(\widehat{\xi_*}(D), \widehat{Q}, D)$. As $\widehat{Q}$ is an SPC query, by the proof of Theorem 35(2), we have that $\mathsf{F}_{\mathsf{rel}}(\widehat{\xi_*}(D), \widehat{Q}, D) \geqslant \frac{1}{1+(\mu_C(\widehat{Q})+2\mu_v(\widehat{Q}))\cdot \max_{\varphi\in\mathcal{A}_0} \bar{d}^m_\varphi(k^*)}$.

(3) Note that when $\xi_*(D) \neq \emptyset$, $d' \neq +\infty$. Thus $\eta \neq 0$. □

Observe the following. Given any RA query Q on a dataset D, (1) $\mathsf{parQP}_{\mathsf{RA}}$ and $\mathsf{instQP}_{\mathsf{RA}}$ compute (a) approximate answers $\xi_*(D)$, and (b) an accuracy bound η, by accessing no more than $\alpha|D|$ tuples in the entire process. (2) The relevance of $\xi_*(D)$ is guaranteed above a non-zero bound, decided by selection conditions in Q and access templates used in ξ_*. (3) When $\xi_*(D) \neq \emptyset$, both the coverage of $\xi_*(D)$ and the accuracy bound η are warranted non-zero.

Correctness & Complexity. The correctness of $\mathsf{instQP_{RA}}$ is warranted by Theorem 38. It takes $\mathsf{instQP_{RA}}$ (a) $O(|Q|\|\mathcal{A}\| + |Q|\|Q\|\lceil\log_2 \alpha|D|\rceil)$ time to generate the α-bounded execution plan ξ_*, the same as $\mathsf{instQP_{SPC}}$; and (b) $O(|S||S'|)$ time to rectify the lower bound η if Q contains set difference.

4.6 Approximating Aggregate Queries

We next extend resource-bounded approximation to aggregate queries. Below we first extend the C-measure for aggregates and group-by (Section 4.6.1). We then give an approximation scheme for aggregate queries (Section 4.6.2).

4.6.1 RC-measure Extended for Aggregate Queries

We consider $\mathsf{RA_{aggr}}$ [Elm08], 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 $\mathsf{RA_{aggr}}$ query itself, (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 aggregate functions max, min, avg, sum or count. The output of Q is a relation of schema R_Q, consisting of attributes V and X. Written in SQL, Q is

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

Since Q' is also an $\mathsf{RA_{aggr}}$ query (embedded in C), $\mathsf{RA_{aggr}}$ supports "nested" aggregate queries. The notions of α-bounded (execution) plans can be readily extended to $\mathsf{RA_{aggr}}$.

RC measure for $\mathsf{RA_{aggr}}$. We extend the distance functions of Section 4.2 to $\mathsf{RA_{aggr}}$, for a set S of approximate answers.

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

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

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

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

- $\delta_{\mathsf{rel}}(Q, s) = \delta_{\mathsf{rel}}(\pi_X(Q'), s[X])$ if there is no $s' \in S$ such that $s \neq s'$ and $s[X] = s'[X]$, and it is $+\infty$ otherwise.

- $\delta_{\mathsf{cov}}(Q,t,s) = \mathsf{max}(\delta_{\mathsf{cov}}(Q',t[X],s[X]), f_{\mathsf{agg}}(t[V],s[V]))$.

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

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 $\delta_{\mathsf{rel}}(Q,s)$ is determined by $\delta_{\mathsf{rel}}(\pi_X(Q'),s[X])$, which measures how "qualified" s is *w.r.t.* the conditions in Q. For coverage, we measure how well $s[X]$ covers $t[X]$ via $\delta_{\mathsf{cov}}(Q',t[X],s[X])$, and how close $s[V]$ is to $t[V]$ via $f_{\mathsf{agg}}()$.

Given the d functions $\delta_{\mathsf{rel}}(Q,s)$ and $\delta_{\mathsf{cov}}(Q,t,s)$, the relevance $\mathsf{F}_{\mathsf{rel}}(S,Q,D)$ and the coverage $\mathsf{F}_{\mathsf{cov}}(S,Q,D)$ are defined in the same way as in Section 4.2.2.

4.6.2 Approximation of Aggregates with Group-by

We next extend the approximation scheme to $\mathsf{RA}_{\mathsf{aggr}}$ under the access schema $\mathcal{A}_0$ of Theorem 29. More specifically, we extend $\mathsf{parQP}_{\mathsf{RA}}$ and $\mathsf{instQP}_{\mathsf{RA}}$ of Section 4.5 to $\mathsf{parQP}_{\mathsf{agg}}$ and $\mathsf{instQP}_{\mathsf{agg}}$, respectively. We start with max and min, and then outline how to handle other aggregate functions.

Algorithm $\mathsf{parQP}_{\mathsf{agg}}$. Given an $\mathsf{RA}_{\mathsf{aggr}}$ query Q with max or min, $\mathsf{parQP}_{\mathsf{agg}}$ generates a canonical plan $\xi_Q = (\xi_Q^F, \xi_Q^E)$.

Fetching plan ξ_Q^F. It is the same as $\mathsf{parQP}_{\mathsf{RA}}$ except that it treats each $\mathsf{gpBy}(Q',X,\mathsf{agg}(V))$ in a max SPC sub-query as $\pi_{X\cup\{V\}}(Q')$, to fetch all attribute values necessary for Q_s.

Evaluation plan ξ_Q^E. It s $\mathsf{parQP}_{\mathsf{RA}}$ as follows. When Q is $\mathsf{gpBy}(Q', X, \mathsf{agg}(V))$, it defines $\mathsf{E}(Q) = \mathsf{gpBy}(\mathsf{E}(Q'), X, \mathsf{agg}(V))$. The rest remains the same as $\mathsf{parQP}_{\mathsf{RA}}$.

Algorithm $\mathsf{instQP}_{\mathsf{agg}}$. It differs from $\mathsf{instQP}_{\mathsf{agg}}$ only in the lower bound function L, which is d as follows. When $Q = \mathsf{gpBy}(\mathsf{E}(Q'), X, \mathsf{agg}(V))$, $d_{\mathsf{rel}}(Q) = d_{\mathsf{rel}}(Q')$ and $d_{\mathsf{cov}}(Q) = d_{\mathsf{cov}}(Q')$. Given the d L, it generates an α-bounded execution plan ξ_* and accuracy bound η, as $\mathsf{instQP}_{\mathsf{agg}}$ does.

Corollary 40: *For* ξ_* *and* η *generated by* $\mathsf{instQP}_{\mathsf{agg}}$, *the lower bounds on* ξ_* *and* η *specified in Theorem 38 remain the same for any* $\mathsf{RA}_{\mathsf{aggr}}$ *query with* min *and* max. □

Proof: By the definition of C-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 answers. Since $\mathsf{parQP}_{\mathsf{agg}}$ enforces the **group by** semantics in terms of the evaluation plan. Thus, the accuracy bounds for RA in Theorem 38 remain the same for $\mathsf{RA}_{\mathsf{aggr}}$ with min and max. □

Remark. When it comes to sum, avg and count, we extend access constraints and templates to keep track of the number of duplicated attribute values, as follows.

<u>Extended access constraints</u>. For each access constraint $\phi = R(X \to Y, N)$, we extend it as follows:

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

where $\overline{\mathsf{occ}}$ is an extended attribute of R for XY. A database instance D of R satisfies ϕ' if

- it satisfies ϕ; and moreover
- there is 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.

<u>Extended access templates</u>. Similarly, for each access template $\varphi(k) = R(X \to Y, k, \bar{d}_Y(k))$, we extend it to

$$\varphi(k)' = R(X \to Y, k, \bar{d}_Y(k), \overline{\mathsf{occ}}(k)),$$

such that a database instance D satisfies $\varphi(k)$ if

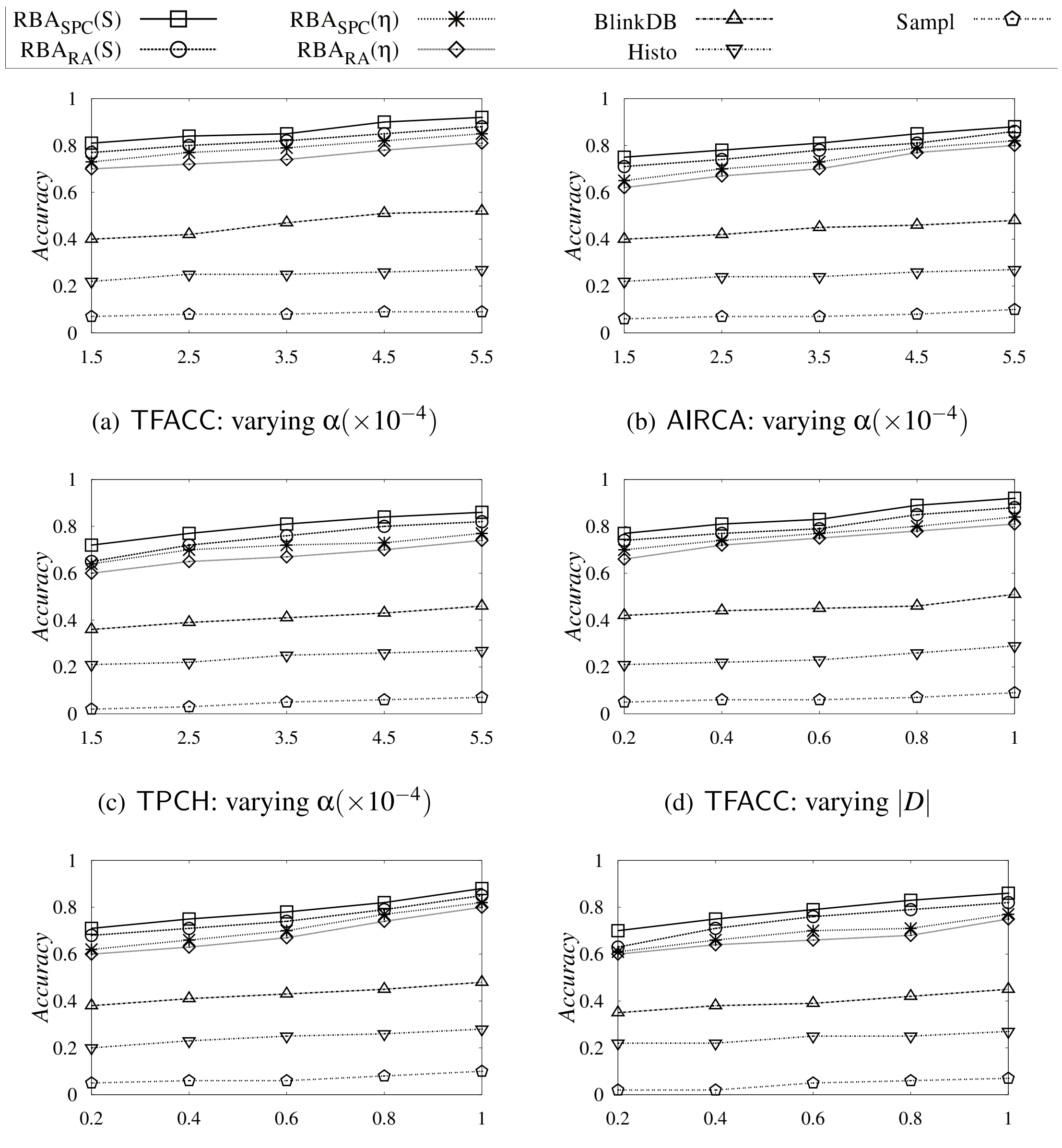
- D satisfies $\varphi(k)$; and
- there exists an index on D that, for each k and each X-value $\bar{a}$, returns 2^k values in $D_{XY}(X = \bar{a})$, $\bar{d}_Y(k)$ as usual, and an approximate occurrence count for each value $\overline{ab}$ returned, which is the sum of occurrences of all values in $D_{XY}(X = \bar{a})$ represented by $\overline{ab}$.

Intuitively, in a template $R(X \to Y, k, \bar{d}_Y(k))$, 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 the extended access schema, $\mathsf{parQP}_{\mathsf{RA}}$ and $\mathsf{instQP}_{\mathsf{RA}}$ can be readily extended to approximate such $\mathsf{RA}_{\mathsf{aggr}}$ queries

Note that the extended access templates for group-by aggregate queries with count, sum and avg work as well as the access template for RA queries when there are no further selections on the result of the group-by aggregation, which is the common case in practice for aggregation.

4.7 Experimental Study

Using real-life and synthetic data, we conducted two sets of experiments to evaluate the effectiveness and the efficiency of our resource bounded approximation scheme.



(a) TFACC: varying $\alpha(\times 10^{-4})$

(b) AIRCA: varying $\alpha(\times 10^{-4})$

(c) TPCH: varying $\alpha(\times 10^{-4})$

(d) TFACC: varying $|D|$

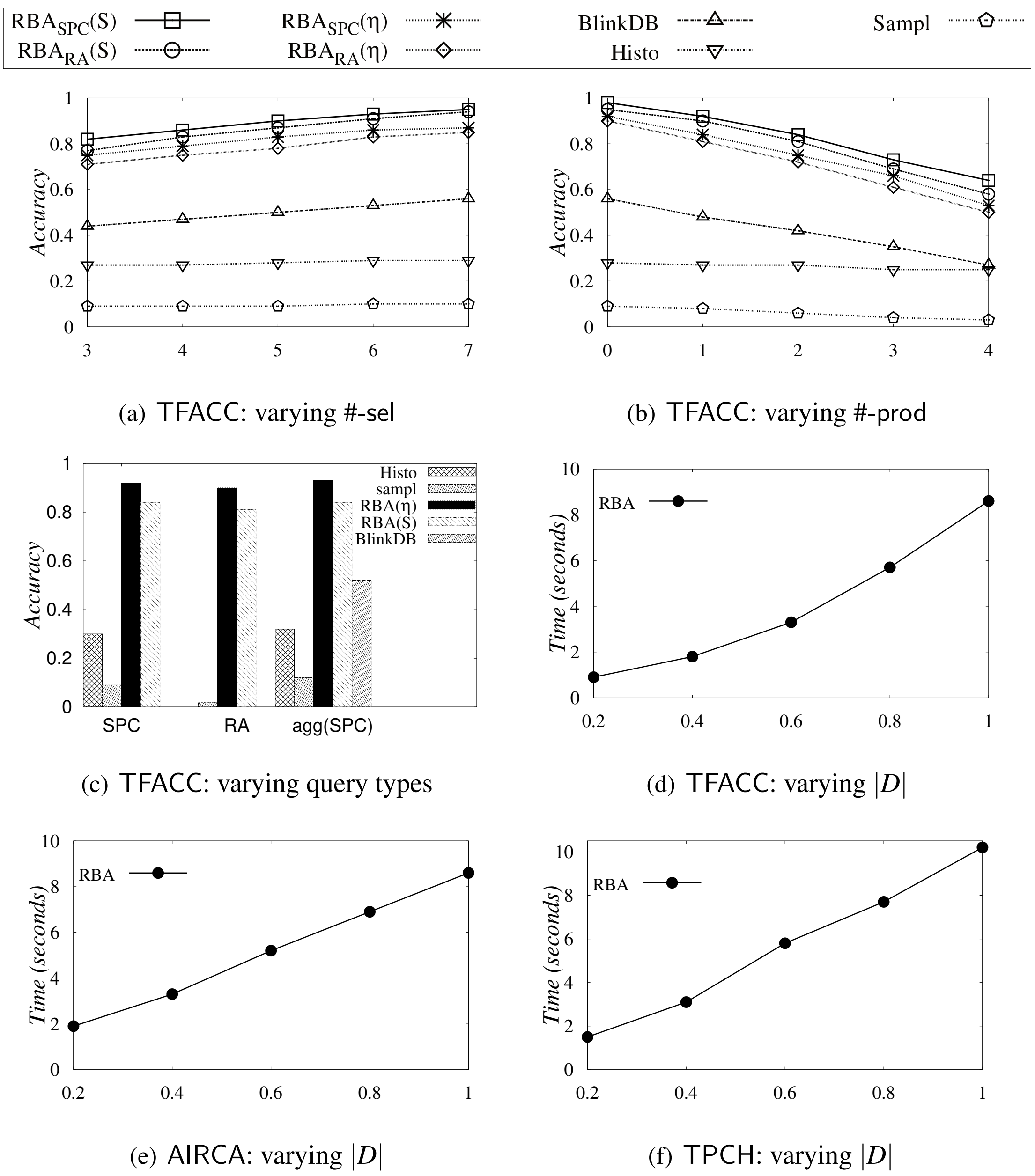
(e) AIRCA: varying $|D|$

(f) TPCH: varying $|D|$

Figure 4.8: Accuracy of resource-bounded approximation scheme

Experimental setting. We used two real-life datasets.

(1) AIRCA integrates Flight On-Time Performance data [BTSa] and Carrier Statistic data [BTSb] for US air carriers from 1987 to 2014. It consists of 7 tables, 358 attributes,



(a) TFACC: varying #-sel
(b) TFACC: varying #-prod
(c) TFACC: varying query types
(d) TFACC: varying $|D|$
(e) AIRCA: varying $|D|$
(f) TPCH: varying $|D|$

Figure 4.9: Accuracy and scalability of resource-bounded approximation scheme

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 [Gova], and National Public Transport Access Nodes [Govb]. It has 19 tables with 113 attributes, and over 89.7 million tuples in total, about 21.4GB of data.

(3) Synthetic data (TPCH) was generated by TPC-H dbgen [TPC] using scale factor 25, yielding 200 million tuples.

Access schema. We discovered 7, 12 and 9 access constraints for AIRCA, TFACC and TPCH, respectively, following [CF16]. We extended the constraints with 24, 31, 20 access templates of the form specified in the proof of Theorem 29(1-2). For each access constraint $R(X \to Y, N)$, we build its index on instance D of R by first projecting D on XY via $\pi_{XY}(D)$, and then building hash index on X attributes. In practice, this can be readily implemented by using any DBMS alone. For each access template $R(X \to Y, k, \bar{d}_Y(k))$, we build its index as follows. (a) For each X-value $\bar{a}$ in D of R, we build a K-D tree $T_{XY}(X = \bar{a})$ on $\sigma_{X=\bar{a}}\pi_{XY}(D)$. (b) For each k, we use the tuples in the $(k+1)$-th level of $T_{XY}(X = \bar{a})$, and set $\bar{d}_Y(k)[A]$ to be the largest widths of the nodes in the $(k+1)$-th level of $T_{XY}(X = \bar{a})$. Here the width of nodes in a K-D tree can be computed by a linear time bottom-up scan of the tree. In practice, this can be implemented by using any programming language together with DBMS. It should be remarked that there are other alternative implementations, *e.g.,* group-by sampling with DBMS via the native languages supported, *e.g.,* PL/pgSQL on postgreSQL.

Queries. We generated 30 queries Q for each dataset, among which 30% are aggregate SPC queries with 1-2 group-by aggregates for each; the others are RA queries, each with 0-3 set differences. 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.

Algorithms. We implemented the following: (1) our resource bounded approximation scheme RBA for RA (aggregate or not), based on the algorithms of Sections 4.4, 4.5 and 4.6; (2) Sampl, an extension of the algorithm of [AGPR99] that samples $\alpha|D|$ tuples and answers queries using the sample with DBMS; (3) Histo, which creates multi-dimensional histograms of size $\alpha|D|$ and uses it to answer queries, following [IP99].

We also compared with (4) BlinkDB, which supports aggregate SPC queries with restricted joins [AMP+13]. We installed BlinkDB (Alpha 0.2.0). For each access constraint (resp. template) $R(X \to Y, N)$ (resp. $R(X \to Y, k, \bar{d}_Y(k))$), we created a sample for BlinkDB by using "**create** table R_sample **as select distinct** X, Y **from** R c", where $c \in [1, \frac{2^k}{\|R[XY]\|}]$, $\|R[XY]\|$ is the number of tuples in $\pi_{XY}(R)$. As such, we provided BlinkDB with the indices of our access schema as samples. Although BlinkDB claims to support evaluation time and accessed data constraints [AMP+13], we could not configure these following its available document. Hence, we manually controlled these via the size of available sample tables.

We evaluated each method using all queries it supports: (a) RA for RBA, aggregate or not, (b) SPC for Histo, aggregate or not, and (c) restricted aggregate SPC for BlinkDB. To compare with Histo and BlinkDB, we also tested SPC for RBA.

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). All the experiments were run 3 times. The average is reported.

Experimental results. We next report our findings.

Exp-1: Effectiveness of resource-bounded scheme.

(1) Accuracy. We first evaluated the accuracy (RC-measure) of approximate answers computed by these methods, by varying resource ratio α, the size $|D|$ of datasets D, and the complexity of queries Q. For SPC, we denote by (a) $\mathsf{RBA}_{\mathsf{SPC}}(S)$ the C-ratio of the approximate answers S computed by RBA, *i.e.,* min $(\mathsf{F}_{\mathsf{rel}}(S,Q,D)$, $\mathsf{F}_{\mathsf{cov}}(S,Q,D))$, and (b) $\mathsf{RBA}_{\mathsf{SPC}}(\eta)$ the computed lower bound η by RBA; similarly for $\mathsf{RBA}_{\mathsf{RA}}(S)$ and $\mathsf{RBA}_{\mathsf{RA}}(\eta)$ for RA. We used full datasets in the experiments unless stated otherwise.

(a) Varying α. Using the full datasets, we varied α from 1.5×10^{-4} to 5.5×10^{-4} on all the datasets, and evaluated the average accuracy of approximate answers computed by all the methods. As reported in Figures 4.8(a), 4.8(b) and 4.8(c), (1) the approximate answers computed by RBA are accurate: $\mathsf{RBA}_{\mathsf{SPC}}(S)$ and $\mathsf{RBA}_{\mathsf{RA}}(S)$ are consistently above 0.72; $\mathsf{RBA}_{\mathsf{SPC}}(S)$ is above 0.85 when $\alpha \geqslant 3.5\times10^{-4}$, 4.5×10^{-4}, and 5.5×10^{-4} on TFACC, AIRCA and TPCH, respectively. (2) RBA outperforms Sampl, Histo and BlinkDB; *e.g.,* when α is 1.5×10^{-4} on TFACC, on average $\mathsf{RBA}_{\mathsf{RA}}$ is 11.6, 3.7 and 2.0 times more accurate than Sampl, Histo and BlinkDB, respectively, and the gap for $\mathsf{RBA}_{\mathsf{SPC}}$ is even larger; the results on the other datasets are similar. (3) The larger α is, the more accurate RBA is. This is because when RBA generates α-bounded plans, it can inspect more tuples *guided by input query Q* (algorithm $\mathsf{instQP}_{\mathsf{agg}}$) to pick those most relevant to Q; in contrast, Sampl and Histo use one-size-fit-all synopses and their accuracy is indifferent to α. While BlinkDB may select samples, we find that it does not work well when there are joins in a query. These verify the *benefit of dynamic data reduction* of RBA. (4) The deterministic bound η estimated by RBA is accurate, *e.g.,* when $\mathsf{RBA}_{\mathsf{SPC}}(S)$ is 0.92 on TFACC with $\alpha = 5.5\times10^{-4}$, $\mathsf{RBA}_{\mathsf{SPC}}(\eta)$ is 0.85.

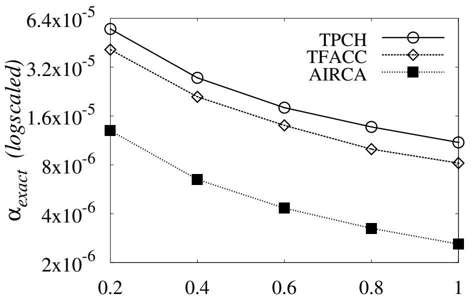
(b) Varying $|D|$. Fixing $\alpha = 5.5\times10^{-4}$, we varied $|D|$ with scale factor σ from 0.2 to 1. As shown in Figures 4.8(d), 4.8(e) and 4.8(f), (1) $\mathsf{RBA}_{\mathsf{SPC}}$ and $\mathsf{RBA}_{\mathsf{RA}}$ have accuracy above 0.75 all the time. (2) RBA performs better than Sampl, Histo and BlinkDB

in accuracy. (3) RBA does substantially *better on larger datasets*: $\mathsf{RBA}_{\mathsf{SPC}}(S)$ and $\mathsf{RBA}_{\mathsf{RA}}(S)$ are above 0.82 when $\sigma \geqslant 0.8$, whereas Sampl, Histo and BlinkDB are not very sensitive to $|D|$. This is because when D grows, RBA makes use of the larger budget $\alpha|D|$ to find tuples more relevant to input query Q, and hence improve S. In contrast, Sampl and Histo does not benefit much from larger D, since C-measure is estimated in terms of the total distances of the values in approximate answers and those in D, and such distances do not necessarily get smaller in larger synopses; similarly for BlinkDB due to its restricted sample selection.

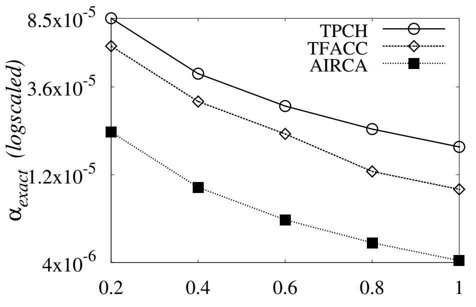
(c) Varying Q. Using full datasets and fixing $\alpha = 5.5 \times 10^{-4}$, we varied #-sel, #-prod and the types of queries Q to evaluate the impact of Q. The results on TFACC are shown in Figures 4.9(a), 4.9(b) and 4.9(c), which are consistent with the results on AIRCA and TPCH (not shown). We treat the accuracy of Histo for RA as 0 since it does not support RA; similarly for BlinkDB when evaluating SPC and RA queries.

We find the following. (1) RBA does better with larger #-sel, *e.g.,* $\mathsf{RBA}_{\mathsf{SPC}}(\eta)$ increases from 0.82 to 0.95 when #-sel changes from 3 to 7. This is because RBA derives and instantiates query plans guided by their relevance to Q. 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 is not as much as RBA. (2) RBA and BlinkDB perform worse with larger #-prod since Cartesian products scale up the total distances of values, as expected (see Section 4.2.1). Histo and Sampl also degrade with larger #-prod for the same reason; but they are less sensitive since their accuracy is mostly dominated by the synopses. (3) RBA performs much better than Histo, Sampl and BlinkDB, and does the best for SPC. In particular, for aggregate SPC supported by all methods (with limited joins to favor BlinkDB), RBA has much higher accuracy than the others: the accuracy of RBA is 0.93 as opposed to 0.52, 0.32 and 0.12 for BlinkDB, Histo and Sampl, respectively.

(2) Resource ratio for exact answer. Varying $|D|$ as in (1b), we evaluated the average α_{exact} for RBA to find exact answers, *i.e.,* with accuracy = 1, for those queries that are answered exactly in (1b). The results for SPC and RA are reported in Figures 4.10(a) and 4.10(b), respectively. We find that the larger $|D|$ is, the smaller α_{exact} is. On full AIRCA, α_{exact} is 2.6×10^{-6} and 4.1×10^{-6} for SPC and RA, respectively. Indeed, (i) the majority of queries that are answered exactly in (1b) are boundedly evaluable. Their plans access a bounded amount of data independent of $|D|$, which is typically very



(a) SPC: varying $|D|$



(b) RA: varying $|D|$

Figure 4.10: Resource ratios for exact answers

	AIRCA	TFACC	TPCH
total index for access schema ($\times|D|$)	5.7x	8.8x	6.5x
index for constraints only ($\times|D|$)	0.01x	0.07x	0.02x

Table 4.2: Size of indices for access schemas

small. (ii) For those that are not bounded, RBA applies access constraints as much as possible, and the remaining parts amount for a small set of D for guided search via access templates.

(3) Index size. We also examined the index size for access schemas. As reported in Table 4.2, the indices for access constraints take less than 7% of the size of the dataset D in all three cases, and the entire indices with templates are in $c|D|$ for $5.7 < c < 8.8$. This is in the range of typical database indices, which account for 3-5 times of $|D|$ for TPC-H [Lev].

Exp-2: Efficiency and scalability. Finally, we evaluated (i) the efficiency of RBA for generating α-bounded plans; and (ii) the scalability of the plans. For all queries on all three full datasets, RBA generates α-bounded plans in less than 200ms. In the same setting of Exp-1(1b), we report the scalability of the plans. As shown in Figures 4.9(d), 4.9(e) and 4.9(f), the plans scales 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 PostgreSQL coult not finish within 3 hours.

Summary. From the experiments we find that the resource-bounded approximation schema RBA allows us to accurately evaluate unpredictable SPC and RA queries, aggregate or not, under small resource ratio α. (1) Its accuracy is consistently above 0.85 when $\alpha \geqslant 5.5 \times 10^{-4}$ on all three datasets for SPC queries, and is 0.82 for RA queries, aggregate or not. (2) RBA is efficient and scalable. It generates α-bounded plans in 200ms, and the plans compute answers for all queries within 10.2 seconds, even on AIRCA of 60GB and TPCH with 200 million tuples, while conventional DBMS cannot complete the jobs within 3 hours. (3) The bound η estimated by RBA is accurate. While Histo, Sampl and BlinkDB are effective when answering simple aggregate queries, they are not as accurate as RBA for full (aggregate) SPC and RA, especially on queries with multiple selections or joins.

Summary

The work is a first step towards striking a balance between available resources and accuracy. We have proposed a framework for querying (big) relations with bounded resources, under a uniform notion of access schema. It novelty consists of the framework, a new accuracy measure, a resource-bounded approximation scheme, and a set of algorithms to support dynamic data reduction. Our experimental study has verified that the framework is promising for answering unpredictable queries, aggregate or not, with deterministic accuracy guarantees under the constraint of a given resource ratio.

Chapter 5

Bounded Evaluability with Views

This chapter presents another approach that extends bounded evaluability to queries that are not boundedly evaluable, by bounded rewriting using views. Under an access schema $\mathcal{A}$, a query Q has a *bounded rewriting* Q' using a set $\mathcal{V}$ of views if for each database D that satisfies $\mathcal{A}$, there exists a fraction D_Q of D such that $Q(D) = Q'(D_Q, \mathcal{V}(D))$, and the size of D_Q and the time for identifying D_Q are determined by query Q and constraints in $\mathcal{A}$ only, independent of the size of D. That is, we can compute $Q(D)$ by accessing cached views and a bounded amount of data in D, no matter how big D is. Here Q, Q' and $\mathcal{V}$ are defined in the same query language.

In this chapter, we study the problem for deciding whether a query has a bounded rewriting given a set $\mathcal{V}$ of views and a set $\mathcal{A}$ of access constraints. We establish the complexity of the problem for various query languages, from Σ_3^p-complete for conjunctive queries (CQ), to undecidable for relational algebra (FO). To explore effective use of bounded rewriting, we develop

- a PTIME effective syntax for FO queries that have a bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$, with a PTIME oracle for checking whether an FO query has bounded output under $\mathcal{A}$; and
- a linear-time effective syntax for FO queries that have bounded output under $\mathcal{A}$.

In this chapter, we propose another approach for queries that are not boundedly evaluable, by incorporating bounded evaluability with views.

Making use of views is an approach that has proven effective by practitioners [ALK+13]. The idea is to select and materialize a set $\mathcal{V}$ of views, and answer Q in a dataset D by using views $\mathcal{V}(D)$ and an additional small fraction of D. That is, we cache $\mathcal{V}(D)$ with fast access, and compute $Q(D)$ by using $\mathcal{V}(D)$ and by restricting costly I/O operations to (possibly big) D. Many queries that are not boundedly evaluable can be efficiently answered using views [ALK+13].

Example 21: Consider a Graph Search query Q_0: *find movies that were released by Universal Studios in 2014, liked by people at NASA, and were rated 5.* The query is defined over a relational schema $\mathcal{R}_0$ consisting of:

- $\mathsf{person}(\mathsf{pid}, \mathsf{name}, \mathsf{affiliation})$,
- $\mathsf{movie}(\mathsf{mid}, \mathsf{mname}, \mathsf{studio}, \mathsf{release})$,
- $\mathsf{rating}(\mathsf{mid}, \mathsf{rank})$ for ranks of movies, and
- $\mathsf{like}(\mathsf{pid}, \mathsf{id}, \mathsf{type})$, indicating that person pid likes item id of type, including but not limited to movies.

Over $\mathcal{R}_0$, Q_0 is written as a conjunctive query:

$$\begin{aligned} Q_0(\mathsf{mid}) = \exists x_p, x'_p, y_m \big(&\mathsf{person}(x_p, x'_p, \text{“NASA”}) \wedge \\ &\mathsf{movie}(\mathsf{mid}, y_m, \text{“Universal”}, \text{“2014”}) \wedge \\ &\mathsf{like}(x_p, \mathsf{mid}, \text{“movie”}) \wedge \mathsf{rating}(\mathsf{mid}, 5)\big). \end{aligned}$$

Consider a set $\mathcal{A}_0$ of two access constraints: (a) $\varphi_1 = \mathsf{movie}(\mathsf{studio}, \mathsf{release} \to \mathsf{mid}, N_0)$, stating that each studio releases at most N_0 movies each year, where N_0 ($\leqslant 100$) is the maximum number of films released by a studio in a single year, which is obtained by aggregating $\mathcal{R}_0$ instances; an index is built on movie such that given any $(\mathsf{studio}, \mathsf{release})$ value, it returns (at most N_0) corresponding mids; and (b) $\varphi_2 = \mathsf{rating}(\mathsf{mid} \to \mathsf{rank}, 1)$, stating that each movie has a unique rating; an index is built on rating to fetch rank as above.

Under $\mathcal{A}_0$, query Q_0 is not boundedly evaluable: an instance D_0 of $\mathcal{R}_0$ may have billions of person and like tuples [GBDS14], and no constraints in $\mathcal{A}_0$ can help us identify a bounded fraction of these tuples to answer Q_0.

Nonetheless, suppose that we are given a view as follows that collects movies liked by NASA folks:

$$V_1(\mathsf{mid}) = \exists x_p, x'_p, y'_m, z_1, z_2 \big(\mathsf{person}(x_p, x'_p, \text{“NASA”}) \wedge$$

$$\mathsf{movie}(\mathsf{mid}, y'_m, z_1, z_2) \wedge \mathsf{like}(x_p, \mathsf{mid}, \text{“movie”})\big).$$

As will be seen later, Q_0 can be rewritten into a conjunctive query Q_ξ using V_1, such that for all instances D_0 of $\mathcal{R}$ that satisfy $\mathcal{A}_0$, $Q_0(D_0)$ can be computed by Q_ξ that accesses only $V_1(D_0)$ and an additional $2N_0$ tuples from D_0, no matter how big D_0 is. Here $V_1(D_0)$ is a small set, although its size is dependent on D_0. □

To support scale independence using views, practitioners have developed techniques for selecting views, indexing the views for fast access and for incrementally maintaining the views [ALK+13]. However, there are still fundamental issues that call for a full treatment. How should we characterize scale independence using views? What is the complexity for deciding whether a query is scale independent given a set of views and access constraints? If the complexity of the problem is high, is there any systematic way that helps us make effective use of cached views for querying big data?

Overview. This chapter tackles these questions.

(1) Bounded rewriting. We formalize scale independence using views (Section 5.1). Consider a query language $\mathcal{L}$, a set $\mathcal{V}$ of $\mathcal{L}$-definable views and a database schema $\mathcal{R}$. Informally, under a set $\mathcal{A}$ of access constraints, we say that a query $Q \in \mathcal{L}$ has a *bounded rewriting* Q' in $\mathcal{L}$ using $\mathcal{V}$ if for each instance D of $\mathcal{R}$ that satisfies $\mathcal{A}$, there exists a fraction D_Q of D such that

- $Q(D) = Q'(D_Q, \mathcal{V}(D))$, and
- the time for identifying D_Q and hence the size $|D_Q|$ of D_Q are independent of the size $|D|$ of D.

That is, we compute the exact answers $Q(D)$ via Q' by accessing cached $\mathcal{V}(D)$ and a *bounded* fraction D_Q of D. While $\mathcal{V}(D)$ may not be bounded, we often use small views cached with fast access [ALK+13]. We formalize the notion in terms of query plans in a form of query trees commonly used in database systems [RG00], which have a bounded size M determined by our available resources such as processors and time constraint.

(2) Complexity. We study *the bounded rewriting problem* (Section 5.2), referred to as $\mathsf{VBRP}(\mathcal{L})$ for a query language $\mathcal{L}$. Given a set $\mathcal{A}$ of access constraints, a query $Q \in \mathcal{L}$ and a set $\mathcal{V}$ of $\mathcal{L}$-definable views, all defined on the same database schema $\mathcal{R}$, and a bound M, $\mathsf{VBRP}(\mathcal{L})$ is to decide whether under $\mathcal{A}$, Q has a bounded rewriting in $\mathcal{L}$ using $\mathcal{V}$ with a query plan no larger than M. The need for studying this problem is evident: if Q has a bounded rewriting, then we can find an efficient query plan to answer Q on possibly big D.

We study $\mathsf{VBRP}(\mathcal{L})$ when $\mathcal{L}$ ranges over conjunctive queries (CQ, *i.e.,* SPC), unions of conjunctive queries (UCQ, *i.e.,* SPCU), positive FO queries ($\exists\mathsf{FO}^+$, select-project-join-union queries) and first-order logic queries (FO, the full relational algebra). We show that VBRP is Σ_3^p-complete for CQ, UCQ and $\exists\mathsf{FO}^+$; but it becomes undecidable for FO. We explore the impact of various parameters ($\mathcal{R}$, M, $\mathcal{A}$ and $\mathcal{V}$) of VBRP on its complexity.

(3) Effective syntax. To cope with the undecidability of $\mathsf{VBRP}(\mathsf{FO})$ and the intractability of $\mathsf{VBRP}(\mathsf{CQ})$, we study effective syntax for FO queries with a bounded rewriting (Section 5.3). Here due to the need for checking output boundedness and the existence of bound M on the query plans, the development of effective syntax is more involved than Chpater 3. Instead of identifying one class of queries, we need two steps. We first develop a PTIME effective syntax for the class of FO queries that have bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$, with a PTIME*oracle* $O_{\mathcal{A}}$ for checking whether FO queries have bounded output under $\mathcal{A}$. More specifically, given any relational schema $\mathcal{R}$, $\mathcal{A}$ of access constraints over $\mathcal{R}$, a set $\mathcal{V}$ of views and a bound M, we identify a class of FO queries, referred to as *queries topped* by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, such that under $\mathcal{A}$,

(a) every FO query that has a bounded rewriting using $\mathcal{V}$ is equivalent to a query topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$;
(b) every query topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ has a bounded rewriting in FO using $\mathcal{V}$; and
(c) it takes PTIME in $|Q|$ in $M, |Q|, |\mathcal{V}|$ and $|\mathcal{A}|$ to syntactically check whether Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, by making use of the oracle $O_{\mathcal{A}}$.

That is, provided an oracle $O_{\mathcal{A}}$, topped queries make a *core* subclass of FO queries with a bounded rewriting using $\mathcal{V}$, without sacrificing their expressive power, along the same lines as rang-safe queries for safe relational calculus (see, *e.g.,* [AHV95]). This allows us to reduce VBRP to syntactic checking of topped queries. and make practical use of bounded rewriting. Indeed, given a query Q, we can check syntactically whether Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ in PTIME, by condition (c) above; if so, we can find a bounded rewriting as warranted by condition (b); moreover, if Q has a bounded rewriting, then it can be transformed to a topped query by condition (a).

We then develop another linear-time effective syntax for the class of FO queries that have bounded output under $\mathcal{A}$, which serves as a "realization" of the oracle $O_{\mathcal{A}}$ used above. More specifically, for any access schema $\mathcal{A}$ over relational schema $\mathcal{R}$, we identify a class $\mathcal{L}_{BO}(\mathcal{R}, \mathcal{A})$ of queries over $\mathcal{R}$, such that

(a) each FO query that has bounded output under $\mathcal{A}$ is $\mathcal{A}$-equivalent to a query in

$\mathcal{L}_{BO}(\mathcal{R},\mathcal{A})$;

(b) each FO query in $\mathcal{L}_{BO}(\mathcal{R},\mathcal{A})$ has bounded output under $\mathcal{A}$; and

(c) it takes $O(|Q|)$ time to check whether an FO query Q is in $\mathcal{L}_{BO}(\mathcal{R},\mathcal{A})$.

Note that we do not fix any parameter of $\mathcal{R}$, M, $\mathcal{A}$ and $\mathcal{V}$ here.

This chapter is an effort to give a formal treatment of scale independence with views, an approach that has already been put in action by practitioners. The complexity bounds reveal the inherent difficulty of the problem. The effective syntax, however, suggests a promising direction for making effective use of bounded rewriting.

5.1 Bounded Query Rewriting

In this section we formalize bounded query rewriting using views under access constraints. Recall the notions of database schema, access schema (access constraints), and query classes of interest from Chpater 1. Below we first revise the notion of bounded query plans under access constraints to better incorporate views.

Query plans. Following [RG00], we define evaluation plans for a query Q using a set $\mathcal{V}$ of views, both defined over a relational schema $\mathcal{R}$. To simplify the definition of query plans, we write Q in the relational algebra in terms of projection π, selection σ, Cartesian product $\times$, union $\cup$, set difference $\setminus$ and renaming ρ.

A *query plan* for Q using $\mathcal{V}$, denoted by $\xi(Q,\mathcal{V},\mathcal{R})$, is a tree T_ξ that satisfies the two conditions below.

(1) Each node u of T_ξ is labeled $S_i = \delta_i$, where S_i denotes a relation for partial results, and δ_i is as follows:

(a) $\{c\}$ for a constant in Q, if u is a leaf of T_ξ;

(b) a view V for $V \in \mathcal{V}$, if u is a leaf of T_ξ;

(c) $\mathsf{fetch}(X \in S_j, R, Y)$, if u has a single child v labeled with $S_j = \delta_j$, and S_j has attributes X;

(d) $\pi_Y(S_j)$, $\sigma_C(S_j)$ or $\rho(S_j)$, if u has a single child v labeled with $S_j = \delta_j$; here Y is a set of attributes in S_j, and C is a condition defined on S_j; or

(e) $S_j \times S_l$, $S_j \cup S_l$ or $S_j \setminus S_l$, if u has two children v and v' labeled with $S_j = \delta_j$ and $S_l = \delta_l$, respectively.

Intuitively, given an instance D of $\mathcal{R}$, relations S_i's are computed by δ_i, bottom up in T_ξ as usual [RG00]. More specifically, δ_i may (a) extract constants from Q, (b) access

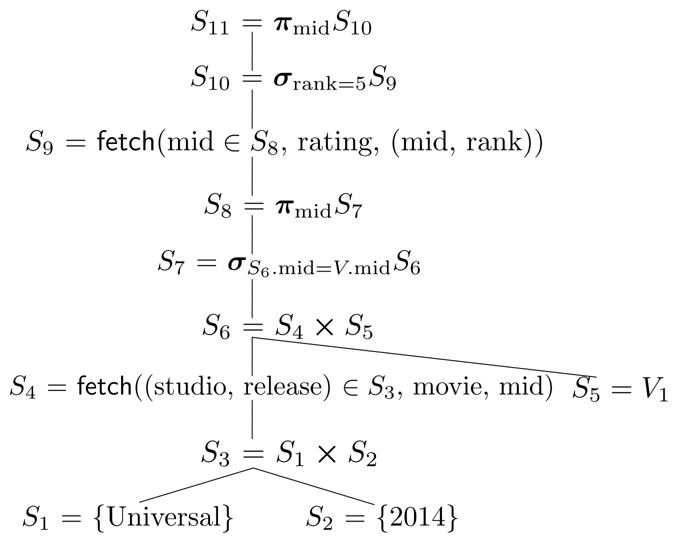

$S_{11} = \pi_{\text{mid}} S_{10}$
$S_{10} = \sigma_{\text{rank}=5} S_9$
$S_9 = \text{fetch}(\text{mid} \in S_8, \text{rating}, (\text{mid}, \text{rank}))$
$S_8 = \pi_{\text{mid}} S_7$
$S_7 = \sigma_{S_6.\text{mid}=V.\text{mid}} S_6$
$S_6 = S_4 \times S_5$
$S_4 = \text{fetch}((\text{studio}, \text{release}) \in S_3, \text{movie}, \text{mid})$
$S_5 = V_1$
$S_3 = S_1 \times S_2$
$S_1 = \{\text{Universal}\}$
$S_2 = \{2014\}$

Figure 5.1: A query plan ξ_0 for Q_0 using view V_1

cached views $V(D)$, and (c) access D via a fetch operation, which, for each $\bar{a} \in S_j$, retrieves $D_{XY}(X = \bar{a})$ from D; it may also be a relational operation ((d) and (e) above). Relation S_n associated with the root of T_ξ is the result of the computation.

(2) For each instance D of $\mathcal{R}$, the *result* $\xi(D)$ of applying $\xi(Q, \mathcal{V}, \mathcal{R})$ to D is the relation S_n at root of T_ξ computed as above. We require that $\xi(D) = Q(D)$.

The *size* of plan ξ is the number of nodes in T_ξ.

Example 22: A plan $\xi_0(Q_0, V, \mathcal{R}_0)$ for Q_0 using view V_1 given in Example 21 is depicted in Fig. 5.1. Given an instance D of $\mathcal{R}_0$, (a) it fetches the set S_4 of mids of all movies released by Universal in 2014, using constants in Q_0; (b) filters S_4 with mids in $V_1(D)$ via join, to get a subset S_8 of S_4 of movies liked by NASA folks; (c) fetches rating tuples using the mids of S_8; and finally, (d) finds the set S_{11} of mids that is precisely $Q_0(D)$. □

Bounded plans. To formalize bounded query rewriting, we bring into play access schema and a bound on the size of query plans. We use the following notions. Consider an access schema $\mathcal{A}$ defined over $\mathcal{R}$.

A plan $\xi(Q, \mathcal{V}, \mathcal{R})$ is said to *conform to* $\mathcal{A}$ if

(a) for each fetch$(X \in S_j, R, Y)$ operation in ξ, there exists an access constraint $R(X \to Y', N)$ in $\mathcal{A}$ such that $Y \subseteq X \cup Y'$, and

(b) there is a constant N_Q such that for all instances D of $\mathcal{R}$ that satisfy $\mathcal{A}$, $|D_Q| \leqslant N_Q$, where D_Q is the set of all tuples fetched for computing $\xi(D)$.

That is, while ξ can access entire cached views, its access to the underlying D is via fetch operations only, by making use of the indices in the access constraints of $\mathcal{A}$. Plan ξ tells us how to retrieve a set D_Q of tuples from D such that $Q(D)$ is computed by using the data in D_Q and $\mathcal{V}(D)$ only. Better still, D_Q is *bounded*: $|D_Q|$ is independent of possibly big $|D|$. The time for identifying and fetching D_Q is also independent of $|D|$ (assuming that given an X-value $\bar{a}$, it takes $O(N)$ time to fetch $D_{XY}(X = \bar{a})$ in D with the index in $R(X \to Y, N)$).

Given a natural number M, we say that $\xi(Q, \mathcal{V}, \mathcal{R})$ is an *M-bounded plan for Q using $\mathcal{V}$ under $\mathcal{A}$* if (a) ξ conforms to $\mathcal{A}$, and (b) the size of ξ is at most M.

Intuitively, M is a threshold picked by users and is determined by available resources. The less resources we have, the smaller M we can afford. If $\xi(Q, \mathcal{V}, \mathcal{R})$ is M-bounded under $\mathcal{A}$, then for all datasets D that satisfy $\mathcal{A}$, we can efficiently answer Q in D by following ξ and accessing a bounded amount of data from D.

Remark. Here we adopt size bounded tree plans instead of the unbounded datalog-style sequential plans in Chapter 2 for the following reasons. (1) Traditional query plans are also defined in terms of algebra trees. (2) The complexity of bounded evaluability will dominate the complexity of bounded rewriting, *i.e.,* EXPSPACE-hard, if we adopt the original definition of bounded plans in Chapter 2. This is caused by both the datalog-style definition of plans and the absence of size constraints on the plans. Therefore, we use size bounded tree plans in this chapter, such that we can explore the essential hardness of bounded rewriting instead of bounded evaluability again.

Example 23: Plan ξ_0 shown in Fig. 5.1 is 11-bounded for Q_0 using V_1 under $\mathcal{A}_0$. Indeed, (a) both fetch operations (S_4 and S_9) are controlled by the access constraints of $\mathcal{A}_0$, and (b) for any instance D of $\mathcal{R}_0$, ξ_0 accesses at most $2N_0$ tuples from D, where N_0 is the constant in φ_1 of $\mathcal{A}_0$, since $|S_4| \leqslant N_0$ by φ_1, and $|S_9| \leqslant N_0$ by $S_8 \subseteq S_4$ and constraint φ_2 on rating in $\mathcal{A}_0$; and (c) eleven operations are conducted in total. Note that rating tuples in D are fetched by using S_8, which is obtained by relational operations on $V_1(D)$ and S_4. While V_1 is not boundedly evaluable under $\mathcal{A}_0$, the amount of data fetched from D is independent of $|D|$. □

Bounded query rewriting. We now formalize this notion. Consider a query Q in a language $\mathcal{L}$, a set $\mathcal{V}$ of $\mathcal{L}$-definable views, and an access schema $\mathcal{A}$, all over the same database schema $\mathcal{R}$. For a bound M, we say that Q has an *M-bounded rewriting in $\mathcal{L}$ using $\mathcal{V}$ under $\mathcal{A}$*, or simply *a bounded rewriting using $\mathcal{V}$* when M and $\mathcal{A}$ are clear from the context, if it has an M-bounded query plan $\xi(Q, \mathcal{V}, \mathcal{R})$ under $\mathcal{A}$ such that ξ is *a query plan in $\mathcal{L}$, i.e.,* in each label $S_i = \delta_i$ of ξ,

symbols	notations
$\mathcal{R}$, R	database schema $\mathcal{R}$ and relation schema $R \in \mathcal{R}$
$\mathcal{A}$	access schema
$D \models \mathcal{A}$	an instance D of $\mathcal{R}$ satisfies access schema $\mathcal{A}$
$Q \in \mathcal{L}$	query Q in a query language $\mathcal{L}$
$\mathcal{V}$, V	a set $\mathcal{V}$ of views and a view $V \in \mathcal{V}$
$\xi(Q, \mathcal{V}, \mathcal{R})$	a query plan ξ for Q using $\mathcal{V}$ over instances of $\mathcal{R}$
T_ξ	plan ξ represented as a query tree
$\xi(D)$	the result of applying ξ to D
$\mathsf{VBRP}(\mathcal{L})$	the bounded rewriting problem for queries in $\mathcal{L}$
$Q \equiv_{\mathcal{A}} Q'$	$\mathcal{A}$-equivalence
$Q \sqsubseteq_{\mathcal{A}} Q'$	$\mathcal{A}$-containment
QP_Q	the set of all possible query plans of a bounded size
$\xi \sqsubseteq_{\mathcal{A}} Q$	$Q_\xi \sqsubseteq_A Q$, query Q_ξ expressed by ξ

Table 5.1: Notations in Chapter 5

- if $\mathcal{L}$ is CQ, then δ_i is a fetch, π, σ, $\times$ or ρ operation;
- if $\mathcal{L}$ is UCQ, δ_i can be fetch, π, σ, $\times$, ρ or $\cup$, and moreover, for any node labeled $\cup$, all its ancestors in the tree T_ξ of ξ are also labeled with $\cup$; that is, $\cup$ is conducted at "the top level" only;
- if $\mathcal{L}$ is $\exists\mathsf{FO}^+$, then δ_i is fetch, π, σ, $\times$, $\cup$ or ρ; and
- if $\mathcal{L}$ is FO, δ_i can be fetch, π, σ, $\times$, $\cup$, $\setminus$ or ρ

One can verify that if ξ is a query plan in $\mathcal{L}$, then there exists a query Q_ξ in $\mathcal{L}$ such that for all instances D of $\mathcal{R}$, $\xi(D) = Q_\xi(D)$ and moreover, $|Q_\xi|$ is linear in the size of ξ. Such query Q_ξ is unique up to equivalence. We refer to Q_ξ as the query *expressed by* ξ.

Example 24: The CQ Q_0 of Example 21 has an 11-bounded rewriting in CQ using V_1 under $\mathcal{A}_0$. Indeed, ξ_0 of Fig. 5.1 is such a bounded plan, which expresses

$$\begin{aligned} Q_\xi(\mathsf{mid}) = \exists y_m \big(&\mathsf{movie}(\mathsf{mid}, y_m, \text{"Universal"}, \text{"2014"}) \\ &\wedge V_1(\mathsf{mid}) \wedge \mathsf{rating}(\mathsf{mid}, 5)\big). \end{aligned}$$

It is a rewriting of Q_0 using V_1 in CQ. $\square$

All these notations are summarized in Table 5.1.

5.2 The Existence of Bounded Rewriting

To make effective use of bounded rewriting, we need to settle *the bounded rewriting problem*, denoted by $\mathsf{VBRP}(\mathcal{L})$ for a query language $\mathcal{L}$ and stated as follows.

- INPUT: A database schema $\mathcal{R}$, a natural number M (in unary), an access schema $\mathcal{A}$, a query $Q \in \mathcal{L}$ and a set $\mathcal{V}$ of $\mathcal{L}$-definable views all defined on $\mathcal{R}$.
- QUESTION: Under $\mathcal{A}$, does Q have an M-bounded rewriting in $\mathcal{L}$ using $\mathcal{V}$?

No matter how important, $\mathsf{VBRP}(\mathcal{L})$ is nontrivial.

Theorem 41: *Problem* $\mathsf{VBRP}(\mathcal{L})$ *is*

(1) Σ_3^p*-complete when* $\mathcal{L}$ *is* CQ*,* UCQ *or* $\exists\mathsf{FO}^+$*; and*

(2) undecidable when $\mathcal{L}$ *is* FO. □

Below we first reveal the inherent complexity of $\mathsf{VBRP}(\mathcal{L})$ by investigating problems embedded in it, and outline a proof of Theorem 41 for various $\mathcal{L}$ (Section 5.2.1). We then investigate the impact of parameters $\mathcal{R}$, $\mathcal{A}$, $\mathcal{V}$ and M on the complexity of VBRP (Section 5.2.2).

5.2.1 The Bounded Rewriting Problem

To understand where the complexity of $\mathsf{VBRP}(\mathcal{L})$ arises, consider a problem embedded in it. Given an access schema $\mathcal{A}$, a query Q, a set $\mathcal{V}$ of views, and a query plan ξ of length M, it is to decide whether ξ is a bounded plan for Q using $\mathcal{V}$ under $\mathcal{A}$. This requires that we check the following: (a) Is the query Q_ξ expressed by ξ equivalent to Q under $\mathcal{A}$? (b) Does ξ conform to $\mathcal{A}$? None of these questions is trivial. To simplify the discussion, we focus on CQ for examples.

$\mathcal{A}$-equivalence. Consider a database schema $\mathcal{R}$. Under an access schema $\mathcal{A}$ over $\mathcal{R}$, we say that two queries Q_1 and Q_2 defined over $\mathcal{R}$ are *$\mathcal{A}$-equivalent*, denoted by $Q_1 \equiv_{\mathcal{A}} Q_2$, if for all instances D of $\mathcal{R}$ that satisfy $\mathcal{A}$, $Q_1(D) = Q_2(D)$. This is a notion weaker than the conventional notion of query equivalence $Q_1 \equiv Q_2$. The latter is to decide whether for all instances D of $\mathcal{R}$, $Q_1(D) = Q_2(D)$, regardless of whether $D \models \mathcal{A}$. Indeed, if $Q_1 \equiv Q_2$ then $Q_1 \equiv_{\mathcal{A}} Q_2$, but the converse does not hold. It is known that query equivalence for CQ is NP-complete (cf. [AHV95]). In contrast, it has been shown that $\mathcal{A}$-equivalence is harder (unless P = NP).

Lemma 42 (Chapter 2): *Given access schema* $\mathcal{A}$ *and two queries* Q_1 *and* Q_2*, it is* Π_2^p*-complete to decide whether* $Q_1 \equiv_{\mathcal{A}} Q_2$*, for* Q_1 *and* Q_2 *in* CQ*,* UCQ *or* $\exists\mathsf{FO}^+$. □

Coming back to VBRP, for a query plan ξ and a query Q, we need to check whether ξ is a query plan for Q, *i.e.,* whether $Q_\xi \equiv_{\mathcal{A}} Q$, where Q_ξ is the query expressed by ξ. This step is Π_2^p-hard for CQ. It is easy to show that it is undecidable when it comes to FO.

Bounded output. Another complication is introduced by views in $\mathcal{V}$. To decide whether a query plan ξ is bounded for a query Q using $\mathcal{V}$ under $\mathcal{A}$, we need to verify that ξ conforms to $\mathcal{A}$. This *may* require us to check whether a view $V \in \mathcal{V}$ has "bounded output".

Example 25: Consider database schema $\mathcal{R}_0$, query Q_0, and access schema $\mathcal{A}_0$ defined in Example 21.

(a) Suppose that instead of V_1, a CQ view V_2 is given:

$$V_2(\mathsf{pid}) = \exists x'_p\ \mathsf{person}(\mathsf{pid}, x'_p, \text{“}NASA''\text{)}.$$

Given an instance D of $\mathcal{R}_0$, $V_2(D)$ consists of people who work at NASA. Extend $\mathcal{A}_0$ to $\mathcal{A}_1$ by including $\varphi_3 = \mathsf{like}((\mathsf{pid}, \mathsf{id}) \to (\mathsf{pid}, \mathsf{id}, \mathsf{type}), 1)$, *i.e.,* $(\mathsf{pid}, \mathsf{id})$ is a key of relation like. Then Q_0 has a rewriting Q_2 using V_2:

$$\begin{aligned} Q_2(\mathsf{mid}) = \exists x_p, y_m\ \big(&V_2(x_p) \wedge \mathsf{like}(x_p, \mathsf{mid}, \text{“movie”}) \wedge \\ &\mathsf{movie}(\mathsf{mid}, y_m, \text{“Universal”}, \text{“2014”}) \wedge \mathsf{rating}(\mathsf{mid}, 5)\big). \end{aligned}$$

One can verify that Q_2 is a bounded rewriting of Q_0 using V_2 under $\mathcal{A}_1$ iff there exists a constant N_1 such that for all instances D of $\mathcal{R}$, if $D \models \mathcal{A}_1$, then $|V_2(D)| \leqslant N_1$; that is, NASA has at most N_1 employees. For if it holds, then we can extract a set S of at most N_0 mids by leveraging constraint φ_1 of $\mathcal{A}_1$ on movie, and select pairs $(\mathsf{pid}, \mathsf{mid})$ from $V_2(D) \times S$ that are in a tuple $(\mathsf{pid}, \mathsf{mid}, \text{“movie”})$ in the like relation, by making use of φ_3 given above. For each mid that passes the test, we check its rating via the index in φ_2, by accessing at most N_0 tuples in rating. Putting these together, we access at most $N_1 \cdot N_0 + N_0$ tuples from D. Conversely, if the output of $V_2(D)$ is not bounded, then Q does not have a bounded rewriting using V_2 under $\mathcal{A}_1$.

(b) For rewriting some queries, we *do not* have to check whether a view has bounded output. For example, consider a rewriting $Q(x) = Q_3(x) \wedge V_3(x)$ of query Q over a database schema $\mathcal{R}$, where V_3 is a view, and Q_3 has a bounded query plan under an access schema $\mathcal{A}$ and does not use any view. Then Q has a bounded rewriting under $\mathcal{A}$ no matter whether $|V_3(D)|$ is bounded or not for instances D of $\mathcal{R}$. Indeed, all data fetching operations are conducted by Q_3; for each x-value a computed by $Q_3(x)$, we only need to validate whether $a \in V(D)$. The checking involves only cached $V_3(D)$, without accessing D, and hence, $|V_3(D)|$ does not need to be bounded. □

To check whether views have a bounded output when it is necessary, we study *the bounded output problem*, denoted by $\mathsf{BOP}(\mathcal{L})$ and stated as follows:

- ◦ INPUT: A database schema $\mathcal{R}$, an access schema $\mathcal{A}$ and a query $V \in \mathcal{L}$, both defined over $\mathcal{R}$.
- ◦ Question: Is there a constant N such that for all instances D of $\mathcal{R}$, if $D \models \mathcal{A}$ then $|V(D)| \leqslant N$?

The analysis of bounded output is also nontrivial.

Theorem 43: *Problem* $\mathsf{BOP}(\mathcal{L})$ *is*

*(1) co*NP *-complete when* $\mathcal{L}$ *is* CQ, UCQ *or* $\exists\mathsf{FO}^+$*; and*

(2) undecidable when $\mathcal{L}$ *is* FO.

When database schema $\mathcal{R}$ *and access schema* $\mathcal{A}$ *are fixed,* BOP *remains co*NP *-hard for* CQ, UCQ *and* $\exists\mathsf{FO}^+$*, and undecidable for* FO. □

Proof: We first show that BOP is coNP -complete for CQ, UCQ and $\exists\mathsf{FO}^+$, and then prove that it is undecidable for FO.

(1) CQ**,** UCQ **and** $\exists\mathsf{FO}^+$. We show that BOP is coNP -hard for CQ and is in coNP for $\exists\mathsf{FO}^+$. To verify the correctness of the reduction to be given in the lower bound and the algorithm to be used in the upper bound, we first provide a characterization of bounded-output $\exists\mathsf{FO}^+$queries, *i.e.,* queries Q in $\exists\mathsf{FO}^+$for which there exists a constant N such that $|Q(D)| \leqslant N$ for any $D \models \mathcal{A}$. To do this, we first introduce the following notation.

Recall the definition of element queries in the proof of Theorem 4 in Chapter 2.

Note that a query Q in $\exists\mathsf{FO}^+$is equivalent to a UCQ $Q_\vee$, and an element query $Q_e(\bar{x})$ of Q is an element query of a disjunct of $Q_\vee$. Obviously, Q has bounded output if and only if each of its element queries has bounded output. It thus suffices to provide a characterization of bounded output CQ Q that satisfy $\mathcal{A}$.

Let Q be such a CQ. To simplify the discussion, we assume *w.l.o.g.* that relation atoms in Q do not contain constants. Instead, all constants appear in equality conditions of the form $x = a$ for some variable x and constant a. We denote by $\mathsf{cvars}(Q)$ the set of "constant" variables in Q that are (transitively) equal to some constant due to the equality conditions in Q, and by $\mathsf{vars}(Q)$ the set of remaining variables in Q, *i.e.,* those that are not equal to some constant. In the following, a variable is always a non-constant variable, unless specified otherwise.

To state the characterization we also need the notion of covered variables in the proof of Theorem 4 in Chapter 2. Recall that the set of *covered variables of Q under $\mathcal{A}$*, denoted by $\mathsf{cov}(Q,\mathcal{A})$, is inductively as follows:

(1) $\mathsf{cov}_0(Q,\mathcal{A}) := \emptyset$;

(2) For $i > 0$, do the following steps until no further variables in $\mathsf{vars}(Q)$ can be added:

- $\mathsf{cov}_i(Q,\mathcal{A}) := \mathsf{cov}_{i-1}(Q,\mathcal{A})$;
- if there exist an atom $R(\bar{x},\bar{y},\bar{z})$ in Q and an access constraint $R(X \to Y,N)$ in $\mathcal{A}$, where $\bar{x}$ corresponds to X and $\bar{y}$ corresponds to Y, then if all variables in $\bar{x}$ belong to $\mathsf{cov}_{i-1}(Q,\mathcal{A})$ then add all variables in $\bar{y}$ to $\mathsf{cov}_i(Q,\mathcal{A})$, if these are not already in $\mathsf{cov}_i(Q,\mathcal{A})$.

We denote by $\mathsf{cov}(Q,\mathcal{A})$ the result set of the process. Note that $\mathsf{cov}(Q,\mathcal{A})$ consists of (non-constant) variables only. Indeed, constant variables are always bounded (equal to some constant) and hence do not affect the boundedness of a query.

Lemma 44: *A* CQ *query $Q(\bar{v})$ that satisfies $\mathcal{A}$ has bounded output iff all non-constant variables in $\bar{v}$ belong to* $\mathsf{cov}(Q,\mathcal{A})$. $\square$

Proof: ($\Leftarrow$) Let $Q'(\bar{u})$ be the CQ obtained from $Q(\bar{v})$ by removing all existential quantifiers, *i.e.,* $Q(\bar{v}) = \exists\bar{z}\, Q'(\bar{u})$, where $\bar{z}$ consists of all variables (constant and non-constant) in $\bar{u}\setminus\bar{v}$. It is easy to see that $\mathsf{cov}(Q,\mathcal{A}) = \mathsf{cov}(Q',\mathcal{A})$.

We next show, by induction on the computation of $\mathsf{cov}(Q',\mathcal{A})$, that for any variable $x \in \mathsf{cov}(Q',\mathcal{A})$, $Q''_x(x) = \exists\bar{u}\setminus\{x\}\, Q'(\bar{u})$ has bounded output. This implies that $Q(\bar{v})$ has bounded output. Indeed, recall that $Q(\bar{v}) = \exists\bar{z} Q'(\bar{u})$ and constant variables in $\bar{v}$ are trivially bounded.

For the base case, $i = 0$ and $\mathsf{cov}_0(Q',\mathcal{A}) = \emptyset$. Clearly, the sentence $\exists\bar{u}\, Q'(\bar{u}) = \exists\bar{v}\, Q(\bar{v})$ has bounded output. Next, assume that the induction hypothesis holds for any $j \in [0,i-1]$. In other words, for any variable $y \in \mathsf{cov}_{i-1}(Q',\mathcal{A})$, $Q''_y(y) = \exists\bar{u}\setminus\{y\}\, Q'(\bar{u})$ has bounded output. We next show that this also holds for every variable in $\mathsf{cov}_i(Q',\mathcal{A})$.

Let y be a variable in $\mathsf{cov}_i(Q',\mathcal{A}) \setminus \mathsf{cov}_{i-1}(Q',\mathcal{A})$. Suppose that y is added to $\mathsf{cov}_i(Q',\mathcal{A})$ via access constraint $R(X \to Y,N) \in \mathcal{A}$ and atom $R(\bar{x},\bar{y},\bar{z})$ in Q'. Then $y \in \bar{y}$, and any (non-constant) variable $x \in \bar{x}$ must be in $\mathsf{cov}_{i-1}(Q',\mathcal{A})$. From the induction hypothesis we know that $Q''_x(x) = \exists\bar{u}\setminus\{x\}\, Q'(\bar{u})$ has bounded output. That is, there exists a number N_x such that for any instance D satisfying $\mathcal{A}$, $|Q''_x(D)| \leqslant N_x$. Furthermore, since $\exists\bar{u}\setminus\{\bar{x}\}Q'(\bar{u})$ is contained in $Q'''(\bar{x}) = \bigwedge_{x_i\in\bar{x}} Q''_{x_i}(x_i)$ and for any $D \models \mathcal{A}$,

$|Q'''(D)| \leqslant M = \prod_{x_i \in \bar{x}} N_{x_i}$, we can see that $\exists \bar{u} \setminus \{\bar{x}\} Q'(\bar{u})$ also has bounded output. From the definition of access constraints, we can further deduce that $\exists \bar{u} \setminus \{\bar{y}\} Q'(\bar{u})$ when evaluated on D generates at most $M \times N$ tuples. In particular, this holds for $Q''_y(y) = \exists \bar{u} \setminus \{y\} Q'(\bar{u})$ and thus the induction hypothesis also holds for y. Since this argument works for any y in $\mathsf{cov}_i(Q', \mathcal{A}) \setminus \mathsf{cov}_{i-1}(Q', \mathcal{A})$, we can conclude that for any $y \in \mathsf{cov}_i(Q', \mathcal{A})$, $Q''_y(y) = \exists \bar{u} \setminus \{y\}\, Q'(\bar{u})$ has bounded output, as desired.

($\Rightarrow$) Suppose that there exists a (non-constant) variable $v \in \bar{v}$ such that $v \notin \mathsf{cov}(Q, \mathcal{A})$. Since v is a free variable in $Q(\bar{v})$, we have that $v \in \pi_A(Q(T_Q))$, where $(T_Q, \bar{u}_Q)$ is the tableau representation of Q and A is an attribute corresponding to v. We next construct instances D_K of $\mathcal{R}$ for $K > 0$ such that $|\pi_A(Q(T_Q \cup D_K))| > K \times |\pi_A(Q(T_Q))|$. Hence, Q does not have bounded output.

We first illustrate the construction of D_K for $K = 1$. In particular, we let D_1 consist of a copy of T_Q. That is, D_1 is equal to T_Q except that every variable z not in $\mathsf{cov}(Q, \mathcal{A})$ is replaced by a primed copy z'. Note that when considering tableaux, we do not need to differentiate between constant and non-constant variables. Indeed, constant variables correspond to constants in the tableau representation.

Clearly, $\{v, v'\} \subseteq \pi_A(Q(T_Q \cup D_1))$ since $v \notin \mathsf{cov}(Q, \mathcal{A})$ and thus $|\pi_A(Q(T_Q \cup D_1))| > |\pi_A(Q(T_Q))|$ holds. It remains to show that $T_Q \cup D_1$ satisfies $\mathcal{A}$. We show this by contradiction. Suppose that $(T_Q \cup D_1) \not\models R(X \to Y, N)$. This means that there exist $N+1$ tuples $t_1, \ldots, t_{N+1}$ in $T_Q \cup D_1$ such that $t_1[X] = \cdots = t_{N+1}[X]$ but $t_i[Y] \neq t_j[Y]$ for all $i \neq j$, $i, j \in [1, N+1]$.

We distinguish between the following three cases:

- $t_1[X]$ consists of constants and variables in $\mathsf{cov}(Q, \mathcal{A})$. In this case, each $t_i[Y]$ also consists of constants and variables in $\mathsf{cov}(Q, \mathcal{A})$ by the access constraint $R(X \to Y, N)$ and the computation of $\mathsf{cov}(Q, \mathcal{A})$. Hence, there must exist $N+1$ tuples $s_1, \ldots, s_{N+1}$ in T_Q such that $s_i[X \cup Y] = t_i[X \cup Y]$ for $i \in [1, N+1]$. This, however, contradicts the assumption that $T_Q \models \mathcal{A}$.
- $t_1[X]$ consists of constants and variables in T_Q, but at least one of these variables is not in $\mathsf{cov}(Q, \mathcal{A})$. In this case, $t_1, \ldots, t_{N+1}$ are tuples in T_Q. This contradicts again the assumption that $T_Q \models \mathcal{A}$.
- $t_1[X]$ contains a primed copy x' of a variable x in T_Q. In this case, $t_1, \ldots, t_{N+1}$ are tuples in D_1. However, they correspond to $N+1$ tuples $s_1, \ldots, s_{N+1}$ in T_Q, where s_i is obtained from t_i by replacing primed variables with their unprimed versions and leaving the unprimed variables and constants in t_i intact.

Clearly, we have that $s_1[X] = \cdots = s_{N+1}[X]$. Furthermore, $s_i[Y] \neq s_j[Y]$ for $i \neq j$, $i, j \in [1, N+1]$. Indeed, suppose that $s_i[Y] = s_j[Y]$ holds for some i and j. Then also $t_i[Y] = t_j[Y]$ holds, contradicting our choice of tuples $t_1 \ldots, t_{N+1}$. Hence, the tuples $s_1, \ldots, s_{N+1}$ in T_Q are contradicting the assumption that $T_Q \models \mathcal{A}$.

We may thus conclude that $T_Q \cup D_1 \models A$. A similar arguments works when D_K is defined to consist of K distinct copies of T_Q. We then have that $\pi_A(Q(T_Q \cup D_K))$ contains at least K distinct copies of v, and thus $|\pi_A(Q(T_Q \cup D_K))| > K \times |\pi_A(Q(T_Q))|$. As before, $T_Q \cup D_K$ can be shown to satisfy $\mathcal{A}$.

Hence, if $Q(\bar{u})$ has bounded output, then every variable u in $\bar{u}$ must be part of $\mathsf{cov}(Q, \mathcal{A})$. □

Having this characterization at hand, we can check whether a given CQ (UCQ, $\exists$FO$^+$) query has bounded output. To this end, we give a more precise statement of Lemma 44 as follows.

Corollary 45: *For a* CQ *(*UCQ, $\exists$FO$^+$*) query $Q(\bar{x})$ and an access schema $\mathcal{A}$, $Q(\bar{x})$ has bounded output iff for every element query $Q_e(\bar{x}')$ of $Q(\bar{x})$, all (non-constant) variables in $\bar{x}'$ belong to* $\mathsf{cov}(Q_e, \mathcal{A})$. □

We are now ready to show that BOP is coNP -hard for CQ and is in coNP for $\exists$FO$^+$.

Lower bound. We show that BOP is coNP -hard for CQ by reduction from the complement of the SAT problem. The SAT problem is to decide, given a propositional formula $\psi = C_1 \wedge \cdots \wedge C_r$ defined over variables $X = \{x_1, \ldots, x_m\}$, whether there exists a truth assignment for X that satisfies ψ. Here for each $i \in [1, r]$, clause C_i is of the form $\ell_1^i \vee \ell_2^i \vee \ell_3^i$, and for each $j \in [1, 3]$, literal ℓ_j^i is either a variable x_l in X or the negation $\bar{x}_l$ of x_l. It is known to be NP-complete (cf. [GJ79]).

Given an instance ψ of SAT, we define a relational schema $\mathcal{R}$, an access schema $\mathcal{A}$, and a CQ query $Q(w)$ such that $Q(w)$ has bounded output iff ψ is false.

(1) The relational schema $\mathcal{R}$ contains the following two kinds of relation schemas: (a) $R_{01}(A), R_\vee(B, A_1, A_2), R_\wedge(B, A_1, A_2)$, and $R_\neg(A, \bar{A})$ to encode Boolean domain and relations, just as in the proof of Theorem 41; and (b) $R_o(I, X)$ to constrain the output.

(2) The access schema $\mathcal{A}$ contains the following: (a) four access constraints to ensure that valid instances of R_{01}, $R_\vee$, $R_\wedge$, and $R_\neg$ are encoding Boolean domain and relations: $R_{01}(\emptyset \to A, 2)$, $R_\vee(\emptyset \to (B, A_1, A_2), 4))$, $R_\wedge(\emptyset \to (B, A_1, A_2), 4)$,

$R_\neg(\emptyset \to (A,\bar{A}),2)$; in other words, these constraints limit the number of tuples in the corresponding instances (see Figure 5.2); and (b) one access constraint $R_o(I \to X,2)$ to bound the output.

(3) The CQ query Q is defined as follows:

$$Q(w) = \exists \bar{x}, w_1, k \Big(Q_c() \wedge Q_X(\bar{x}) \wedge Q_\psi(\bar{x}, w_1) \wedge R_{01}(w_1)$$
$$\wedge R_o(k,1) \wedge R_o(k,w_1) \wedge R_o(k,w) \Big),$$

where Q_c, Q_X, and Q_ψ are in CQ. The query Q_c is to ensure that the instances of R_{01}, $R_\vee$, $R_\wedge$, and $R_\neg$ contain all tuples shown in Figure 5.2. For example, to include the two tuples in I_{01} in the figure, Q_c will contains two atoms $R_{01}(0)$ and $R_{01}(1)$. Together with the access constraints this implies that whenever $Q(D) \neq \emptyset$ for an instance $D \models \mathcal{A}$, D consists of instances I_{01}, $I_\vee$, $_\wedge$, $I_\neg$ as shown in Figure 5.2, and a non-empty instance I_o of R_o.

Query $Q_X(\bar{x})$ is to ensure that $\bar{x}$ is a truth-assignment of X. From the definition of Q_c and the access constraint $R_{01}(\emptyset \to A,2)$, we know that $Q_X(\bar{x})$ can be defined as $\bigwedge_{1 \leqslant i \leqslant m} R_{01}(x_i)$. Furthermore, query $Q_\psi(\bar{x}, w_1)$ is such defined that when given a truth-assignment μ_X encoded by $\bar{x}$, it sets $w_1 = 1$ if $\psi(\mu_X)$ is true and sets $w_1 = 0$ otherwise. It is easily verified that Q_ψ can encode ψ by leveraging R_{01}, $R_\vee$, $R_\wedge$ and $R_\neg$.

Finally, consider the sub-query $R_o(k,1) \wedge R_o(k,w_1) \wedge R_o(k,w)$. If Q_ψ sets $w_1 = 1$ then we know from $R_o(I \to X,2) \in \mathcal{A}$ that w can be any value. In contrast, if Q_ψ sets $w_1 = 0$, then w can only be 0 or 1. In other words, w is bounded iff $w_1 = 0$.

To verify the correctness of the reduction we leverage Lemmas 45 and 44. More specifically, we show that the variable w is constant in every element query $Q_e(w)$ of $Q(w)$ iff ψ is false. To see this, we need to inspect element queries of $Q(w)$. First, observe that the sub-query $R_{01}(0) \wedge R_{01}(1) \wedge \bigwedge_{1 \leqslant i \leqslant m} R_{01}(x_i)$ violates $R_{01}(\emptyset \to A,2)$, and every element query Q_e of Q must assign each x_i either to 0 or 1. In other words, every element query Q_e encodes a truth assignment μ_X of X. Similarly, due to the access constraints on $R_\vee, R_\wedge$ and $R_\neg$ and the presence of Q_c, in every element query Q_e, Q_ψ correctly evaluates ψ for the truth assignment μ_X encoded in Q_e. Furthermore, in order for w to be in $\mathsf{cov}(Q_e, \mathcal{A})$, observe that $R_o(I \to X,2)$ cannot be used to put w in $\mathsf{cov}(Q_e, \mathcal{A})$, since the variable k cannot be in $\mathsf{cov}(Q_e, \mathcal{A})$ due to the access constraints. However, in Q_e either $R_o(k,1) \wedge R_o(k,w)$ occurs (when $w_1 = 1$) or $R_o(k,1) \wedge R_o(k,0)$ occurs (when $w_1 = 0$). In the latter case, w has become a constant variable; thus Lemma 44 applies and $Q_e(w)$ has bounded output. In the former case, w remains a non-constant variable that is not in $\mathsf{cov}(Q_e, \mathcal{A})$. Hence, when $w_1 = 1$ in Q_e, Q_e is not bounded. Thus $Q_e(w)$ has bounded output iff the truth assignment μ_X encoded in Q_e makes ψ false. As a consequence, Q has bounded output iff ψ is false.

Note that in the reduction above, both $\mathcal{R}$ and $\mathcal{A}$ are fixed, *i.e.,* they do not depend on ψ.

Upper bound. We give an NP algorithm to check the complement of BOP for $\exists\mathsf{FO}^+$. From Lemma 45, we know that given a query $Q(\bar{x})$ in $\exists\mathsf{FO}^+$, to check whether $Q(\bar{x})$ does not have bounded output, we only need to guess an element query $Q_e(\bar{x})$ of Q in which there is a variable x in $\bar{x}$ that does not belong to $\mathsf{cov}(Q_e, \mathcal{A})$. Note that Q is equivalent to a UCQ $Q_\vee$, and an element query $Q_e(\bar{x})$ of Q is an element query of a disjunct of $Q_\vee$. The NP algorithm thus (i) guesses disjunctions in $Q(\bar{x})$ to obtain a CQ query $Q'(\bar{x})$; and (ii) guesses a valuation ν of Q' to constants and variables appearing in Q'. It then verifies whether $\nu(Q') \models \mathcal{A}$ and whether there exists a variable x such that $x \in \nu(\bar{x})$ but $x \notin \mathsf{cov}(\nu(Q'), \mathcal{A})$. It is easy to show that all element queries can be obtained in this way and that computing $\mathsf{cov}(\nu(Q'), \mathcal{A})$ is in PTIME. If the guesses pass this test then we have found a counterexample for Q to be of bounded output. Otherwise, we reject the guess. Hence, this process is in NP and decides whether Q has no bounded output. We thus conclude that deciding BOP is in coNP for $\exists\mathsf{FO}^+$.

(2) FO. We show that BOP is undecidable for FO queries by reduction from the complement of the satisfiability problem for FO queries. Given an FO query Q_1, we define a relational schema $\mathcal{R}$, an access schema $\mathcal{A}$, and an FO query Q, which are the same as their counterparts given in the proof of Theorem 41 for FO. One can easily verify that Q_1 is not satisfiable iff there exists a constant N such that over instances D of $\mathcal{R}$, $|Q(D)| \leqslant N$. Indeed, $Q(x) = R(x) \wedge Q_1()$, where $\mathcal{A} = \emptyset$ and hence, $R(x)$ is not bounded.

As will be seen in the proof of Corollary 47, the satisfiability problem for FO remains undecidable when Q_1 is defined over a fixed relational schema. As a result, BOP(FO) is undecidable when $\mathcal{R}$ and $\mathcal{A}$ are fixed. □

Using Lemma 42 and Theorem 43, we prove Theorem 41.

Proof of Theorem 41. We first study VBRP for CQ, UCQ and $\exists\mathsf{FO}^+$, and then investigate it for FO.

(1) When $\mathcal{L}$ is CQ, UCQ, or $\exists\mathsf{FO}^+$. It suffices to show that VBRP is Σ_3^p-hard for CQ, and is in Σ_3^p for $\exists\mathsf{FO}^+$.

Lower bound. We show that $\mathsf{VBRP}(\mathsf{CQ})$ is Σ_3^p-hard by reduction from the $\exists^*\forall^*\exists^*$3CNF problem, which is known to be Σ_3^p-complete [Sto76]. The $\exists^*\forall^*\exists^*$3CNF problem is to decide, given a sentence $\phi = \exists X \forall Y \exists Z\, \psi(X,Y,Z)$, where $X = \{x_1,\ldots,x_m\}$, $Y = \{y_1,\ldots,y_n\}$, $Z = \{z_1,\ldots,z_p\}$, and ψ is a SAT instance, whether ϕ is true.

Given an instance $\phi = \exists X \forall Y \exists Z\, \psi(X,Y,Z)$, we define a CQ query Q, a set $\mathcal{V}$ of CQ views, an access schema $\mathcal{A}$, and a number M, such that Q has an M-bounded rewriting in CQ using $\mathcal{V}$ under $\mathcal{A}$ iff ϕ is true.

(1) The relational schema $\mathcal{R}$ consists of the following relation schemas: (1) $R_{01}(A)$, $R_\vee(B,A_1,A_2)$, $R_\wedge(B,A_1,A_2)$, and $R_\neg(A,\bar{A})$ are to encode the Boolean domain and operations, as will be explained shortly; (2) $R_Y(I_1,I_2,Y)$ is to store one truth-assignment of Y; (3) $R_o(I,Y)$ is to store a particular tuple, which the query plans can check only via fetch operations; and (4) $R_I(I,K)$ is to store the keys for the relation R_o.

(2) The access schema $\mathcal{A}$ consists of the following: (a) four access constraints to ensure that R_{01}, $R_\vee$, $R_\wedge$ and $R_\neg$ encode Boolean domain and relations: $R_{01}(\emptyset \to A,2), R_\vee(A_1 \to (A_2,B),2), R_\wedge((A_1,A_2) \to B,1)$, and $R_\neg(A \to \bar{A},1)$; (b) an access constraint $R_Y((I_1,I_2) \to Y,1)$ to ensure that we only handle one truth-assignment of Y at a time; and (c) two access constraints $R_o(I \to Y,1)$ and $R_I(I \to K,1)$ for R_o and R_I, respectively, stating that I is a key for R_o and R_I.
It should be noted that the access constraints for $R_\vee$ and $R_\wedge$ are different. In $R_\vee$, we require that when the values corresponding to A_1 are bounded, the values corresponding to A_2 and B are bounded. While in $R_\wedge$, we require that only when both of the values corresponding to A_1 and A_2 are bounded, the values corresponding to B are bounded. This difference is important for our construction.

(3) The query Q is defined as follows:

$$Q() = \exists \bar{y},k\big(Q_c() \wedge Q_Y(\bar{y}) \wedge (\bigwedge_{1\leqslant j\leqslant n} R_Y(j,1,y_j)) \wedge R_I(y_1,k) \wedge R_o(k,1)\big).$$

Here, query Q_c is to ensure that the instances of R_{01}, $R_\vee$, $R_\wedge$ and $R_\neg$ contain all tuples shown in Figure 5.2. It can clearly be expressed in CQ. Query $Q_Y(\bar{y})$ is used to ensure that the values of $\bar{y}$ are Boolean values, and it is defined as $\bigwedge_{1\leqslant i\leqslant n} R_{01}(y_i)$. Note that for $D \models \mathcal{A}$, $Q(D) \neq \emptyset$ implies that the tuples in D corresponding to R_Y encode a valid truth-assignment of Y.

(4) The set $\mathcal{V}$ of CQ views consists of a single view:

$$V(\bar{x},k) = \exists w,\bar{x}',\bar{y},\bar{z}\big(Q_c() \wedge Q_2(w,\bar{x},\bar{x}') \wedge Q_3(w,\bar{y},\bar{z}) \wedge Q_4(\bar{y},w,k)\big)$$

$$\wedge\, Q_5(\bar{x},w) \wedge Q_\psi(\bar{x}',\bar{y},\bar{z},1)\big).$$

Before explaining the component queries in V in more details, we provide some intuition. The view is defined in such a way that if a query plan ξ that uses V does not "fix" the values of $\bar{x}$, then ξ will not conform to $\mathcal{A}$, since the values that k can take will not be bounded. Here, by fixing values we mean that V appears in the query plan in the form of $\sigma_{X=\bar{c}}(V)$, where X are the attributes corresponding to $\bar{x}$ and $\bar{c}$ is a constant tuple. Furthermore, we will see that $\bar{c}$ must consist of Boolean values for $\sigma_{X=\bar{c}}(V)$ to be of use for answering Q. Hence, $\bar{c}$ encodes a truth-assignment of X.

To construct V in this way, we isolate the values of $\bar{x}$ from k by using a new copy $\bar{x}'$ of $\bar{x}$ in component queries. Moreover, we link the possible values for k to those of a variable y_1, and connect the possible values of y_1 to the values that variable w can take. The latter is shown to be unbounded when $\bar{x}$ is not fixed. Hence, when $\bar{x}$ is not fixed, k will be unbounded. We next show how this is achieved by detailing each of the sub-queries in V:

- $Q_c()$ is the same CQ as the one used in Q. For instances $D \models \mathcal{A}$, $Q_c(D)$ evaluates to true iff the instance in D of R_{01} is equal to I_{01}, and the instances in D of $R_\vee$, $R_\wedge$ and $R_\neg$ contain all tuples as listed in Figure 5.2.
- $Q_2(w,\bar{x},\bar{x}') = \bigwedge_{1\leqslant k\leqslant m} R_\wedge(x_i',w,x_i)$. This query is to ensure that if the values of $\bar{x}$ are Boolean, then $\bar{x}'$ and $\bar{x}$ take the same values. Clearly, this only holds when $w=1$. Indeed, in that case due to the access constraint on $R_\wedge$ and the presence of $Q_c()$ in V, we have that for any $D \models \mathcal{A}$, if $Q_c(D) \neq \emptyset$ then $\sigma_{A=1}(Q_2(D))$ consists of tuples of the form $(1,\bar{x},\bar{x})$. Here, A denotes the first attribute in the result schema of Q_2. When either $w=0$ or w and $\bar{x}$ do not take Boolean values, then the access constraint on $R_\wedge$ only imposes a cardinality restriction, and the values in $\bar{x}'$ and $\bar{x}$ are not necessarily the same.
- $Q_3(w,\bar{y},\bar{z}) = \exists \bar{y}',\bar{z}'\left(\bigwedge_{1\leqslant k\leqslant n} R_\vee(y_k',w,y_k) \wedge \bigwedge_{1\leqslant k\leqslant p} R_\vee(z_k',w,z_k)\right)$.
  This query is to ensure that whenever $w=0$ or $w=1$ then the values of $\bar{y}$ and $\bar{z}$ must be Boolean values as well. As before this is due to the presence of $R_\vee(A_1 \to (A_2,B),2)$ and $Q_c()$. In other words, for any $D \models \mathcal{A}$ such that $Q_c(D) \neq \emptyset$, $\sigma_{A=0/1}(Q_3(D))$ consists of tuples of the form $(0/1,\bar{y},\bar{z})$, $\bar{y}$ and $\bar{z}$ are tuples of Boolean values and A denotes the first attribute in the result schema of Q_3. If w can take arbitrary values, however, then the values for $\bar{y}$ and $\bar{z}$ are unconstrained.

$$I_{01} = \begin{array}{|c|} A \\ \hline 1 \\ 0 \end{array} \quad I_{\vee} = \begin{array}{|ccc|} B & A_1 & A_2 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{array} \quad I_{\wedge} = \begin{array}{|ccc|} B & A_1 & A_2 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{array} \quad I_{\neg} = \begin{array}{|cc|} A & \bar{A} \\ \hline 0 & 1 \\ 1 & 0 \end{array}$$

Figure 5.2: Relation instances used in the proof of Theorem 41

– $Q_4(\bar{y}, w, k) = \left(\bigwedge_{1 \leqslant j \leqslant n} R_Y(j, w, y_j) \right) \wedge R_I(y_1, k)$, which is to fetch the truth-assignment of Y and the value of k. As argued before (for Q_3) the values y_j will be Boolean only when $w = 0$ or $w = 1$, and thus only in these cases $Q_4(D) \neq \emptyset$ implies that a truth assignment of Y is embedded in D.

– $Q_\psi(\bar{x}', \bar{y}, \bar{z}, 1)$ is to check whether ψ is true given the values $\bar{x}'$, $\bar{y}$, and $\bar{z}$. The encoding makes use of R_{01}, $R_\vee$, $R_\wedge$ and $R_\neg$. It is only when $\bar{x}'$, $\bar{y}$ and $\bar{z}$ take Boolean values that this query correctly encodes ψ.

– The last query Q_5 is to ensure that if $V(\bar{x}, k)$ is used in a query plan that is relevant for answering Q and conforms to $\mathcal{A}$, then it can only be used when all variables in $\bar{x}$ are assigned a constant Boolean value. Furthermore, when this is the case, w must be 1. As just described, this implies that $\bar{x}' = \bar{x}$, $\bar{y}$ and $\bar{z}$ take Boolean values, and Q_ψ correctly evaluates ψ. It is to encode this that we make use of the difference of the access constraints on $R_\vee$ and $R_\wedge$. Intuitively, the constraint on $R_\vee$ is used to check whether each variable in $\bar{x}$ takes a constant value, since it only takes the attribute A_1 as input. In contrast, since the access constraint on $R_\wedge$ takes both A_1 and A_2 as input, we use it to encode the conjunction of the results of checking each variable in $\bar{x}$. The query Q_5 is defined as follows:

$$Q_5(\bar{x}, w) = \exists \bar{x}'', \bar{v}, \bar{v}', \bar{v}'', \bar{v}''' \left(\bigwedge_{1 \leqslant k \leqslant m} (R_\vee(v_k, x_k, x''_k) \wedge R_\vee(v''_k, v_k, v'_k) \wedge R_\neg(x''_k, v'_k)) \right.$$
$$\left. \wedge R_\wedge(v'''_2, v''_1, v''_2) \wedge \Big(\bigwedge_{2 \leqslant k \leqslant m-2} R_\wedge(v'''_{k+1}, v'''_k, v''_{k+1}) \Big) \wedge R_\wedge(w, v'''_{m-1}, v''_m) \right).$$

To see the effect of this query, consider $D \models \mathcal{A}$ such that $Q_c(D) \neq \emptyset$ and consider $\sigma_{X=\mu_X}(Q_5(D))$, where X are the attributes corresponding to $\bar{x}$, and μ_X is a truth-assignment of X. In this case, the access constraint $R_\vee(A_1 \to (A_2, B), 2)$ ensures that all the values of $\bar{x}''$, $\bar{v}$, $\bar{v}''$, and $\bar{v}'''$ are Boolean values as well. Moreover, $Q_c(D) \neq \emptyset$ ensures that the Boolean operations are correctly encoded in D. Hence, the expression in Q_5 between brackets encodes the truth value of the tautology $\bigwedge_{1\leqslant k\leqslant m}(x_k \vee x''_k \vee \neg x''_k)$. Since the truth value is encoded in w, we have that $w = 1$. In other words, when all $\bar{x}$ values are fixed Boolean values in Q_5, then all previous queries in V work as desired as these required Boolean values for $\bar{x}$ and $w = 1$.

Suppose next that we still fix all $\bar{x}$ values, but not all of them take Boolean values. In this case, Q_5 requires the existence of certain tuples in the instances of $R_\vee$, $R_\wedge$ or $R_\neg$ that are not required by Q. That is, there exists $D \models A$ for which $Q(D) \neq \emptyset$ but $Q_5(D) = \emptyset$ (and thus also $V(D) = \emptyset$). Clearly, using V in this way to answer Q is not very helpful. Hence, when all variables in $\bar{x}$ are fixed, we may assume that these values are Boolean.

It remains to rule out the case when some variables in $\bar{x}$ are not fixed. Suppose that we set all variables in $\bar{x}$ to a Boolean value, except for x_1. Let $X' = X \setminus \{x_1\}$ and consider an instance $D \models \mathcal{A}$ and $\sigma_{X'=\mu_{X'}}(Q_5)(D)$ for some truth-assignment $\mu_{X'}$ of X'. Clearly, the query result contains tuples of the form $(a, \mu_{X'}, w)$ for constants a and w. Since a can be arbitrary, access constraints $R_\vee(A_1 \to (A_2, B), 2)$ only implies that at most two tuples s and t in D exist and are associated to $R_\vee$ such that $s[A_1] = t[A_1] = a$. However, it does not impose any restrictions on the other values in these two tuples. These values can thus be non-Boolean. Similarly, $R_\neg(A \to \bar{A}, 1)$ does not impose value restrictions (except for a cardinality constraint) when $R_\neg(x''_1, v'_1)$ can bind x''_1 and v'_1 with arbitrary values. The same holds for $R_\wedge((A_1, A_2) \to B, 1)$ and $R_\wedge(v'''_2, v''_1, v''_2)$. Although v''_2 takes only Boolean values (recall that we fixed x_2 to a Boolean value), v''_1 can be arbitrary and so can be v'''_2. A similar argument shows that all v'''_i can be arbitrary and so can be w. It should be noted that w can take an arbitrary value for any possible binding of x_1 to the underlying database. Hence, $\sigma_{X'=\mu_{X'}}(Q_5)$ does not have bounded output. We next argue that $\sigma_{X'=\mu_{X'}}(V)$ also does not have bounded output and thus cannot be used in a query plan that conforms to $\mathcal{A}$. Indeed, this readily follows Q_4, which now can bind y_1 with arbitrary values since $R_Y(1, w, y_1)$ can be mapped to various tuples with distinct w-values; and similarly $R_I(y_1, k)$ can be mapped to various tuples resulting in an unbounded number of k values.

In summary, Q_5 ensures that whenever V appears in a query plan that conforms to $\mathcal{A}$, it must have all of its $\bar{x}$ values fixed to some Boolean values.

(5) We set $M = 6$, *i.e.,* only query plan trees with at most six nodes are considered.

To show the correctness of the reduction, we first argue that if Q has an M-bounded rewriting using V under $\mathcal{A}$, then this rewriting can only be of a very specific form. Indeed, since $Q(D)$ depends on the instance D (for some D, $Q(D) = \emptyset$, for others $Q(D) \neq \emptyset$), the query plan cannot be one of the two trivial plans that always return $\emptyset$ or $()$. In other words, the query plan has to use V. Furthermore, since V does not contain R_o, whereas $Q(D)$ depends on the tuples in D corresponding to R_o, the query plan needs to fetch data from R_o. Consider such a fetch operation $\mathsf{fetch}(I \in S_j, R_o, Y)$. We distinguish between the following two cases: (i) S_j is equal to a constant c; or (ii) S_j is the result of some more complex query plan. Note that (i) is not helpful for answering Q as the value k used in the atom $R_o(k,1)$ in Q is arbitrary and may thus be distinct from the constant c. We can therefore assume that we are in case (ii). Moreover, the atom $R_o(k,1)$ in Q asks for a tuple with its second attribute set to 1. This requirement needs to be encoded in the query plan as well, *e.g.,* by means of a constant selection condition $\sigma_{Y=1}$. This selection must occur after the fetch operation. Observe also that since Q is Boolean, whereas the fetch operation, the constant selection, and V are not, the query plan must contain a projection of the form $\pi_\emptyset$. This projection clearly must come last in the query plan (*i.e.,* it is at the root). From this we know that $\mathsf{fetch}(I \in S_j, R_o, Y)$ has at least one selection and projection as ancestor in the query plan tree.

We next analyze the query plan for S_j. We need to consider two options: (a) S_j does have V as a descendant in the query plan tree; and (b) S_j does not have V as a descendant.

In case (a) the query plan for S_j must contain a projection π_A so that S_j is unary. Indeed, recall that R_o is binary and the access constraint takes the first attribute of R_o as input, while V is not unary. Furthermore, as argued above, the only way that V can be used in a query plan that conforms to $\mathcal{A}$ is when it occurs as $\sigma_{X=\mu_X^0}(V)$, *i.e.,* all its $\bar{x}$-values are fixed Boolean values by means of a truth-assignment μ_X^0 of X. This selection condition needs to be accounted for in the query plan. Note also that this constant selection should not be expanded to include the last attribute in V. Indeed, this would make S_i equal to a constant (case (i) above), which is not helpful in answering Q. From this we know that $\mathsf{fetch}(I \in S_j, R_o, Y)$ has at least V, a selection and a projection as descendants. Put together with our earlier observation, these account for the six possible nodes in the query plan. In fact, this completely fixes possible query plans. Indeed, the query plan must be of the form $S_1 = \pi_\emptyset(S_2)$; $S_2 = \sigma_{Y=1}(S_3)$; $S_3 = \mathsf{fetch}(I \in S_4, R_o, Y)$; $S_4 = \pi_A(S_5)$; $S_5 = \sigma_{X=\mu_X^0}(S_6)$ and $S_6 = V$, for some truth-assignment μ_X^0 of X. Furthermore, since we argued that S_4 should not just be a constant value, the projection π_A should be imposed on the last attribute of V (the other ones are fixed by means of the selection condition in S_5).

In case (b), observe that the overall query plan must use V. Here this implies that V must occur in a subtree of the query plan different from the subtree rooted at $\mathsf{fetch}(I \in S_j, R_o, Y)$. At least one node is required to glue these subtrees together. For the query plan for S_j, since S_j is not equal to a constant, we still need to distinguish the following two cases: (b1) S_j is $\mathsf{fetch}(\emptyset, R_{01}, A)$, *i.e.,* the only possible query plan of size 1 that does not use V; (b2) the size of the query plan for S_j is at least 2. For case (b1), similar to case (i) above, we can show that it is not helpful for answering Q. Then we only need to consider case (b2). However, we have at least two nodes in the query plan tree for S_j, one for V, and at least one to glue the subtrees together (as argued above), accounting for four nodes. Combined with the (minimal) three nodes needed for $\mathsf{fetch}(I \in S_j, R_o, Y)$ and its ancestors, this results in a query plan of at least seven nodes, exceeding the bound $M = 6$. Hence, case (b2) cannot occur.

As a consequence, the only possible query plans are of the form as given in case (a).

We may thus conclude that if Q has a 6-bounded query plan using V under $\mathcal{A}$, then this query plan is $\mathcal{A}$-equivalent to $Q'_{\mu_X^0} = \pi_\emptyset\left(\sigma_{\bar{x}=\mu_X^0}(V(\bar{x},k)) \bowtie R_o(k,1)\right)$ for some truth-assignment μ_X^0 of X. We next show that $Q \equiv_\mathcal{A} Q'_{\mu_X^0}$ for some truth-assignment μ_X^0 of X iff ϕ is true. For convenience, we express $Q'_{\mu_X^0}$ as the CQ $Q'_{\mu_X^0} = \exists k\,(V(\mu_X^0,k) \wedge R_o(k,1))$.

($\Leftarrow$) Suppose that ϕ is true and let μ_X^0 be a truth-assignment of X such that $\forall Y \exists Z \psi(\mu_X^0, Y, Z) = \mathsf{true}$. Consider $Q'_{\mu_X^0} = \exists k\,(V(\mu_X^0,k) \wedge R_o(k,1))$ and its unfolding

$$\exists k\Bigg(\exists w,\bar{x}',\bar{y},\bar{z}\Bigg(Q_c() \wedge \bigwedge_{1\leqslant k\leqslant m} R_\wedge(x_i', w, \mu_X^0(x_i)) \wedge \exists \bar{y}',\bar{z}'\Bigg(\bigwedge_{1\leqslant k\leqslant n} R_\vee(y_k', w, y_k) \wedge$$
$$\bigwedge_{1\leqslant k\leqslant p} R_\vee(z_k', w, z_k)\Bigg) \wedge \Bigg(\bigwedge_{1\leqslant j\leqslant n} R_Y(j, w, y_j)\Bigg) \wedge R_I(y_1, k) \wedge Q_5(\mu_X^0, w)$$
$$\wedge\, Q_\psi(\bar{x}', \bar{y}, \bar{z}, 1)\Bigg) \wedge R_o(k,1)\Bigg).$$

Since μ_X^0 is a truth-assignment of X, $Q_5(\mu_X^0, w)$ will assign $w = 1$. As a consequence

$\bar{x}' = \mu_X^0$ and $\bar{y}$ and $\bar{z}$ take Boolean values and the unfolding of $Q'_{\mu_X^0}$ is $\mathcal{A}$-equivalent to

$$\exists k\Bigg(\bar{y},\bar{z}\Bigg(Q_c() \wedge Q_Y(\bar{y}) \wedge Q_Z(\bar{z}) \wedge \Bigg(\bigwedge_{1\leqslant j\leqslant n} R_Y(j,1,y_j)\Bigg) \wedge R_I(y_1,k) \\ \wedge Q_\psi(\mu_X^0,\bar{y},\bar{z},1)\Bigg) \wedge R_o(k,1)\Bigg), \quad (\dagger)$$

where $Q_Y(\bar{y})$ and $Q_Z(\bar{z})$ encode that $\bar{y}$ and $\bar{z}$ take Boolean values, just as in Q.

Consider an instance $D \models \mathcal{A}$ such that $Q(D) \neq \emptyset$. As remarked earlier, this implies that the tuples in D corresponding to R_Y encode a truth assignment μ_Y of Y. Moreover, tuples $(\mu_Y(y_1),k)$ and $(k,1)$ are present in D (for relation R_I and R_o, respectively). Hence, if $Q(D) \neq \emptyset$ then $Q'_{\mu_X^0}(D) \neq \emptyset$ iff $\exists\bar{z}Q_\psi(\mu_X^0,\mu_Y,\bar{z},1)$ evaluates to true. Since $\forall Y\exists Z\psi(\mu_X^0,Y,Z)$ is true, we know that $\exists Z\psi(\mu_X^0,\mu_Y,Z)$ is true. Hence, $Q(D) \neq \emptyset$ implies that $Q'_{\mu_X^0}(D) \neq \emptyset$. In other words, $Q \sqsubseteq_{\mathcal{A}} Q_{\mu_X^0}$. For the converse, $Q_{\mu_X^0} \sqsubseteq_{\mathcal{A}} Q$, observe that when $Q(D) = \emptyset$ then so is $Q'_{\mu_X^0}(D)$. Indeed, the query shown in ($\dagger$) is just like Q but with some additional restrictions ($Q_Z(\bar{z})$ and $Q_\psi(\mu_X^0,\bar{y},\bar{z},1)$). Hence, we may conclude that $Q \equiv_{\mathcal{A}} Q'_{\mu_X^0}$ and thus Q has a 6-bounded query rewriting using V under $\mathcal{A}$.

($\Rightarrow$) Suppose that ϕ is false, yet Q has a 6-bounded rewriting ξ using V under $\mathcal{A}$. As argued before, $\xi \equiv_{\mathcal{A}} Q'_{\mu_X^0}$ for some truth-assignment μ_X^0 of X. Since ϕ is false, there must exist a truth-assignment μ_Y^0 of Y such that $\exists Z\psi(\mu_X^0,\mu_Y^0,Z) = \mathsf{false}$. Let D be a database instance such that $D \models \mathcal{A}$, $Q(D) \neq \emptyset$, and the tuples in D corresponding to R_Y encode the truth assignment μ_Y^0. Since $\exists Z\psi(\mu_X^0,\mu_Y^0,Z) = \mathsf{false}$, we know that $Q_\psi(\mu_X^0,\mu_Y^0,\bar{z},1)(D) = \emptyset$. Then we have that $Q'_{\mu_X^0}(D) = \emptyset$. As a consequence, $Q \not\equiv_A Q'_{\mu_X^0}$. Since this argument works for any truth-assignment μ_X of X, Q is not $\mathcal{A}$-equivalent to any Q'_{μ_X} for μ_X of X. Since these are the only possible 6-bounded rewriting, Q does not have a 6-bounded rewriting using V under $\mathcal{A}$.

Upper bound. We next provide an Σ_3^p algorithm for $\mathsf{VBRP}(\exists\mathsf{FO}^+)$, which works as follows:

(1) guess a query plan ξ such that $|\xi| \leqslant M$;

(2) check whether ξ conforms to $\mathcal{A}$; if not, then return false; otherwise continue;

(3) rewrite ξ into a query Q' in $\exists\mathsf{FO}^+$by substituting the view definition for each view used in ξ;

(4) check whether $Q' \equiv_{\mathcal{A}} Q$. If so, then return true; otherwise, return false.

It is easy to see the correctness of the algorithm. For its complexity, we will show that step (2) can be done in P^{NP}. Moreover, step (3) can be done in PTIME since ξ is a tree, and $|Q'|$ is bounded by $O(|\xi|\cdot|\mathcal{V}|)$. Step (4) requires checking whether $Q' \equiv_{\mathcal{A}} Q$. This was shown to be in Π_2^p (Lemma 42). Putting these together, the algorithm is in Σ_3^p.

Now it remains to show that step (2) can be done in P^{NP}. It suffices to show the following.

Lemma 46: *Given a query plan ξ, it is in P^{NP} to decide whether ξ conforms to $\mathcal{A}$.* □

Proof: To check whether ξ conforms to $\mathcal{A}$, it suffices to verify that for each $\mathsf{fetch}(X \in S_j, R, Y)$ operation in ξ, the following conditions hold: (a) there exists an access constraint $R(X \to Y', N)$ in $\mathcal{A}$ such that $Y \subseteq X \cup Y'$; and (b) there exists a constant N such that for all instances D of $\mathcal{R}$ that satisfy $\mathcal{A}$, $|S_j| \leqslant N$ in the computation of $\xi(D)$, independent of $|D|$.

For each $\mathsf{fetch}(X \in S_j, R, Y)$ operation, it is in PTIME to check condition (a). We use the following algorithm to check condition (b). Let ξ' be the sub-tree of ξ rooted at S_j. The algorithm works as follows.

(1) express ξ' as an equivalent query Q_j in $\exists\mathsf{FO}^+$;

(2) unfold Q_j by replacing each view with its definition, yielding Q'_j in $\exists\mathsf{FO}^+$;

(3) check whether Q'_j has bounded output; if so, return true; otherwise, return false.

The correctness of the algorithm is immediate. For its complexity, observe that steps (1) and (2) are in PTIME, and step (3) is in coNP by Theorem 43. Since there are at most $O(|\xi|)$ fetch operations in ξ, the algorithm is in P^{NP}. □

(2) When $\mathcal{L}$ is FO. We show that VBRP is undecidable for FO queries by reduction from the complement of the satisfiability problem for FO queries, which is undecidable (cf. [AHV95]). The satisfiability problem for FO is to decide, given an FO query Q, whether there exists a database D such that $Q(D) \neq \emptyset$.

Given an FO query Q_1, we construct a relational schema $\mathcal{R}$, an access schema $\mathcal{A}$, an FO query Q, a set $\mathcal{V}$ of FO views, and a natural number M, such that Q has an M-bounded rewriting in FO using $\mathcal{V}$ under $\mathcal{A}$ iff there exists no database D such that $Q_1(D) \neq \emptyset$. More specifically, the construction is as follows: (1) the relational schema $\mathcal{R}$ contains all relation names used by Q_1, and one new unary relation schema $R(X)$; (2) $\mathcal{A} = \emptyset$; (3) query Q is defined as $Q(x) = R(x) \wedge Q_1()$; (4) $\mathcal{V} = \emptyset$; and finally, (5) $M = 1$.

Since $\mathcal{V} = \emptyset$, $\mathcal{A} = \emptyset$, and $M = 1$, the only possible 1-bounded rewriting of Q is the constant query $Q_\emptyset$, which returns $\emptyset$ on all datasets. It is easy to verify that $Q(x) \equiv_{\mathcal{A}} Q_\emptyset$

iff $Q(x) \equiv Q_\emptyset$ iff for any instance D of $\mathcal{R}$, $Q_1(D) = \emptyset$, *i.e.,* when Q_1 is not satisfiable. □

5.2.2 Special Cases

One might be tempted to think that fixing some parameters of VBRP would make our lives easier. In practice we often have predefined database schema $\mathcal{R}$, access schema $\mathcal{A}$, bound M and views $\mathcal{V}$, while queries Q and instances D of $\mathcal{R}$ vary.

Unfortunately, fixing $\mathcal{R}$, $\mathcal{A}$, M and $\mathcal{V}$ does not simplify the analysis of VBRP for FO.

Corollary 47: *There exist fixed $\mathcal{R}$, $\mathcal{A}$, M and $\mathcal{V}$ such that it is undecidable to decide, given an* FO *query Q, whether Q has an M-bounded rewriting in* FO *using the fixed $\mathcal{V}$ under the fixed $\mathcal{A}$.* □

Proof: This is verified by reduction from the complement of the satisfiability problem for FO queries over a fixed relational schema. The latter problem remains undecidable. Indeed, it is verified by reduction from the Post Correspondence Problem, and the reduction uses a database schema consisting of two fixed relation schemas [AHV95]. Hence the reduction to VBRP(FO) given in the proof of Theorem 41 suffices to verify Corollary 47, since it employs fixed $\mathcal{A}$, $\mathcal{V}$ and M only, and $\mathcal{R}$ can also be fixed as argued above. □

We now study the impact of parameters on VBRP for CQ, UCQ and $\exists$FO$^+$. Our main conclusion is that fixing $\mathcal{R}$, $\mathcal{A}$ and M does not simplify the analysis of VBRP. When $\mathcal{V}$ is also fixed, VBRP becomes simpler for these classes of positive queries, but only to an extent.

Fixing $\mathcal{R}$, $\mathcal{A}$ and M. Fixing database schema, access schema and plan size does not help us. Indeed, the Σ_3^p lower bound for CQ is verified by using fixed $\mathcal{R}$, $\mathcal{A}$ and M (Theorem 41). From this the corollary below follows.

Corollary 48: *There exist fixed $\mathcal{R}$, $\mathcal{A}$ and M such that it is Σ_3^p-complete to decide, given a query Q in $\mathcal{L}$ and a set $\mathcal{V}$ of $\mathcal{L}$-definable views over $\mathcal{R}$, whether Q has an M-bounded rewriting in $\mathcal{L}$ using $\mathcal{V}$ under the fixed $\mathcal{A}$, when $\mathcal{L}$ is one of* CQ, UCQ *and* $\exists$FO$^+$. □

5.3 Effective Syntax

The robust undecidability of VBRP for FO and intractability for CQ motivate us to develop effective syntax for FO queries with a bounded rewriting. Below we approach this in two steps. For any schema $\mathcal{R}$, views $\mathcal{V}$, access schema $\mathcal{A}$ and a bound M,

(1) we first develop a PTIME effective syntax $\mathcal{L}_{BR}$ for the class of FO queries that have M-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$, with a PTIME*oracle* $O_{\mathcal{A}}$ for checking whether an FO query has bounded output under $\mathcal{A}$ (Section 5.3.1);

(2) we then develop a PTIME effective syntax $\mathcal{L}_{BO}$ for the class of FO queries that have bounded output under $\mathcal{A}$, as an "efficient realization" of the oracle $O_{\mathcal{A}}$ used in (1) (Section 5.3.2).

5.3.1 An Effective Syntax for Bounded Rewriting with an Oracle

For any $\mathcal{R}$, $\mathcal{V}$, $\mathcal{A}$ and M, we identify a class $\mathcal{L}_{\mathsf{BR}}(\mathcal{R},\mathcal{V},\mathcal{A},M)$ of FO queries, referred to as *queries topped by* $(\mathcal{R},\mathcal{V},\mathcal{A},M)$. We show that it serves as an effective syntax for FO queries with bounded rewriting, with a PTIME oracle $O_{\mathcal{A}}$ for checking queries with bounded output under $\mathcal{A}$.

Theorem 49: *For any $\mathcal{R}$, $\mathcal{V}$, $\mathcal{A}$ and M,*

(a) each FO *query that has an M-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$ is $\mathcal{A}$-equivalent to a query in $\mathcal{L}_{\mathsf{BR}}(\mathcal{R},\mathcal{V},\mathcal{A},M)$;*

(b) every FO *query in $\mathcal{L}_{\mathsf{BR}}(\mathcal{R},\mathcal{V},\mathcal{A},M)$ has an M-bounded rewriting in* FO *using $\mathcal{V}$ under $\mathcal{A}$; and*

(c) it takes PTIME *in $|Q|$ to check whether an* FO *query Q is in $\mathcal{L}_{\mathsf{BR}}(\mathcal{R},\mathcal{V},\mathcal{A},M)$, with a* PTIME *oracle $O_{\mathcal{A}}$ for checking whether Q has bounded output under $\mathcal{A}$.*

Here $\mathcal{A}$, Q and $\mathcal{V}$ are defined over the same $\mathcal{R}$. □

That is, $\mathcal{L}_{\mathsf{BR}}(\mathcal{R},\mathcal{V},\mathcal{A},M)$ is a core sub-class of FO queries with a bounded rewriting. There are possibly other forms of effective syntax for such FO queries, and Theorem 49 just aims to demonstrate the feasibility of "covering" all such FO queries with a class of queries that can be syntactically checked in PTIME with an output boundedness checking oracle. We will handle the oracle in Section 5.3.2. Note that Theorem 49 does not contradict to Corollary 47 due to the requirement of $\mathcal{A}$-equivalence in condition (a) above.

Below we first define the class $\mathcal{L}_{\mathsf{BR}}(\mathcal{R},\mathcal{V},\mathcal{A},M)$. We then give a proof of Theorem 49.

Class $\mathcal{L}_{\mathsf{BR}}(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. A topped query $Q(\bar{z})$ is defined in terms of two functions $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z}))$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$, where $\bar{z}$ is the set of free variables of Q. As will be seen shortly, the parameter $Q_s(\bar{x})$ is to propagate intermediate queries containing views, for Q to assess whether the views have bounded output.

- Boolean function $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z}))$ is true if under $\mathcal{A}$, $Q(\bar{z})$ has a bounded rewriting plan using $\mathcal{V}$, and $Q_s(\bar{x})$ has bounded output; it is false otherwise.
- Function $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ computes a natural number, which is an upper bound on the size of minimum query plans for $Q(\bar{z})$ using $\mathcal{V}$ under $\mathcal{A}$.

We will ensure that if $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$, $Q(\bar{z})$ has a $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$-bounded plan using $\mathcal{V}$ under $\mathcal{A}$.

Using the functions, we now define *topped queries*. An FO query Q over $\mathcal{R}$ is in $\mathcal{L}_{\mathsf{BR}}(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ if

(1) $\mathsf{cov}(Q_\varepsilon, Q) = \mathsf{true}$; and

(2) $\mathsf{size}(Q_\varepsilon, Q) \leqslant M$,

where Q_ε is a "tautology query" such that for any query Q, $Q_\varepsilon \wedge Q = Q$. That is, we compute $\mathsf{cov}(Q_\varepsilon, Q)$ and $\mathsf{size}(Q_\varepsilon, Q)$ starting with $Q_s = Q_\varepsilon$, and conclude that $Q \in \mathcal{L}_{\mathsf{BR}}(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ if the two conditions are satisfied.

Functions $\mathsf{cov}()$ *and* $\mathsf{size}()$. We now provide the details. Consider an FO query Q. To simplify discussion, following [AHV95], we assume *w.l.o.g.* the following: (a) no variable x occurs both free and bound in Q, and there exists at most one quantifier for each x; (b) there exists no universal quantifier in Q, *i.e.,* we replace each $\forall \bar{x}\ \psi(\bar{x})$ with $\neg\exists\bar{x}(\neg\psi(\bar{x}))$; (c) only variables occur in relation atoms, *e.g.,* $R(x,1)$ is replaced with $\exists y\ (R(x,y) \wedge y = 1)$; and (d) no relation atoms contain repeated variables, *e.g.,* we substitute $R(x,y) \wedge x = y$ for each $R(x,x)$.

We define $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z}))$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ inductively based on the structure of Q, separated into eight cases. In the following, we set $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{false}$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = +\infty$ if $Q_s(\bar{x})$ does not have bounded output under $\mathcal{A}$. We assume that variables $\bar{x}$ correspond to attributes X of a relation atom $R(X,Y)$ that occurs in Q; similarly for $\bar{y}$ and Y, $\bar{z}$ and Z.

(1) $Q(\bar{z})$ is $R(\bar{z})$. We have three cases to consider:

(a) if $R(\emptyset \to Z, N) \in \mathcal{A}$, then $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = 1$; otherwise

(b) if $R(X \to U, N) \in \mathcal{A}$ with $X \neq \emptyset$ and $X \cup U = Z$, and $\mathsf{cov}(Q_\varepsilon, Q_s(\bar{x})) = \mathsf{cov}(Q_s(\bar{x}), Q_\varepsilon) = \mathsf{true}$, then $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) =$

$\mathsf{size}(Q_\varepsilon, Q_s(\bar{x})) + 1$; otherwise

(c) $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{false}$, $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = +\infty$.

When Q is a relation atom $R(\bar{z})$, Q has a bounded plan if either (a) $R(\bar{z})$ is covered by a constraint $R(\emptyset \to Z, N)$ in $\mathcal{A}$; or (b) there exists $R(X \to U, N)$ in $\mathcal{A}$ with $X \cup U = Z$, such that we can use the bounded output $\bar{x}$-values of $Q_s(\bar{x})$ to fetch $\bar{z}$-values from instances of R.

Intuitively, Q_s is checked in case (b) for bounded output. It happens when R appears in a query Q', and Q_s is processed in a query plan for Q' prior to R, possibly by operating on views in $\mathcal{V}$. Since $\mathsf{cov}(Q_\varepsilon, Q_s(\bar{x})) = \mathsf{cov}(Q_s(\bar{x}), Q_\varepsilon) = \mathsf{true}$, $Q_s(\bar{x})$ has bounded output and a bounded plan. Hence, we can use its $\bar{x}$ values to instantiate attributes X of R, and fetch data from instances of R using the X-value and the index for $R(X \to U, N)$. More specifically, let ξ_{Q_s} be a plan of size K for $Q_s(\bar{x})$; then a plan for Q is $(T = \xi_{Q_s}, \mathsf{fetch}(X \in T, R, U))$, of size $K + 1$. This is how $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ is derived for case (b). The size of case (a) is immediate.

(2) $Q(\bar{z})$ is $z = c$. For equality atom with a constant, $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = 1$.

(3) $Q(\bar{z})$ is $V(\bar{z})$. We can access cached views; thus, $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = 1$.

(4) $Q(\bar{z})$ is $Q'(\bar{z}) \wedge (x' = y')$. We define

- $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{cov}(Q_s(\bar{x}), Q'(\bar{z}))$, and
- $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ as $\mathsf{size}(Q_s(\bar{x}), Q'(\bar{z})) + 1$ when $\mathsf{cov}(Q_s(\bar{x}), Q'(\bar{z})) = \mathsf{true}$, and as $+\infty$ otherwise.

Given a bounded plan ξ' for Q', a bounded plan for Q is $(T = \xi', \sigma_{X'=Y'}(T))$, increasing the size of ξ' by 1.

(5) $Q(\bar{z})$ is $\exists \bar{z}' Q'(\bar{z}', \bar{z})$. There are two cases.

(a) if $Q'(\bar{z}', \bar{z})$ is a relation atom $R(\bar{z}', \bar{z})$ and there exists $R(\emptyset \to Z, N)$ in $\mathcal{A}$, then $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$, and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = 1$; otherwise

(b) $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{cov}(Q_s(\bar{x}), Q'(\bar{z}', \bar{z}))$; here $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ is $\mathsf{size}(Q_s(\bar{x}), Q'(\bar{z}', \bar{z})) + 1$ if $\mathsf{cov}(Q_s(\bar{x}), Q'(\bar{z}', \bar{z})) = \mathsf{true}$, and is $+\infty$ otherwise.

When Q' is a relation atom $R(Z, Z')$, $\exists \bar{z}' Q'(\bar{z}', \bar{z})$ differs from case (1) above in that it computes the projection of $R(Z, Z')$ on Z. Now if Q' is covered by $R(\emptyset \to Z, N)$, then Q has a bounded plan $\mathsf{fetch}(X \in \emptyset, R, Z)$. When Q' is not a relation atom, if Q' has a bounded plan ξ' of size K, then Q has a plan $(T = \xi', \pi_Z(T))$ of size $K + 1$. Otherwise,

Q' may not have a bounded plan, and we cannot ensure that Q has a bounded plan.

(6) $Q(\bar{z})$ is $Q_1(\bar{z}) \vee Q_2(\bar{z})$. Let μ_i = $\mathsf{cov}(Q_s(\bar{x}),\ Q_i(\bar{z}))$, and s_i = $\mathsf{size}(Q_s(\bar{x}),\ Q_i(\bar{z}))$ for $i \in [1,2]$. Then

(a) if both μ_1 and μ_2 are true, then $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z}))$ = true, and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = $s_1 + s_2 + 1$; otherwise

(b) $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z}))$ = false, $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = $+\infty$.

Intuitively, if Q_1 and Q_2 have bounded plans ξ_1 and ξ_2, respectively, then $Q(\bar{z})$ has a bounded plan $(T_1 = \xi_1, T_2 = \xi_2, T_1 \cup T_2)$ of size bounded by $|\xi_1| + |\xi_2| + 1 \leqslant$ $\mathsf{size}(Q_s(\bar{x}), Q_1(\bar{z}))$ + $\mathsf{size}(Q_s(\bar{x}), Q_2(\bar{z}))$ + 1.

(7) $Q(\bar{z})$ is $Q_1(\bar{x}_1) \wedge Q_2(\bar{x}_2)$, where $\bar{z} = \bar{x}_1 \cup \bar{x}_2$. Let μ_i = $\mathsf{cov}(Q_s(\bar{x}),\ Q_i(\bar{x}_i))$ and s_i = $\mathsf{size}(Q_s(\bar{x}),\ Q_i(\bar{x}_i))$ for $i \in [1,2]$, and μ' = $\mathsf{cov}(Q_s(\bar{x}) \wedge Q_1(\bar{x}_1), Q_\varepsilon)$. Then

(a) if μ_1 = true and μ_2 = true, then $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z}))$ = true, $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = $s_1 + s_2 + \lambda$, where $\lambda = 0$ if Q_1 or Q_2 is of the form $(x = c)$, otherwise $\lambda = 1$ if $\bar{x}_1 \cap \bar{x}_2$ = $\emptyset$, and $\lambda = 4$ if $\bar{x}_1 \cap \bar{x}_2 \neq \emptyset$; otherwise

(b) if μ_1 = true, $Q_2(\bar{x}_2)$ is of the form $\exists \bar{w} R(\bar{x}_1, \bar{x}_2', \bar{w})$, and $R(X_1 \to U, N)$ is in $\mathcal{A}$ with $X_1 \cup X_2' = X_1 \cup U = X_2$, then $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z}))$ = μ'; $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ is $\mathsf{size}(Q_s(\bar{x}), Q_1(\bar{x}_1))$ + 1 if μ' = true, and is $+\infty$ if μ' = false; otherwise

(c) $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z}))$ = false, $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = $+\infty$.

Note that Q_s is expanded in case (b) above. As illustrated in Example 25, the need for testing bounded output arises when we use $\mathsf{fetch}(X \in S_j, R, Y)$ and when S_j is extracted from some view in $\mathcal{V}(D)$; that is, when the view is used in conjunction with fetch. Hence we expand Q_s here, and the expanded Q_s will be propagated to other cases that take Q as a sub-query.

In addition, observe the following.

(a) If both Q_1 and Q_2 have bounded plans, *e.g.*, ξ_1 and ξ_2, respectively, then Q also has a bounded plan, whose size depends on the forms of $Q_1(\bar{x}_1)$ and $Q_2(\bar{x}_2)$, as reflected by different values of λ. More specifically, if, say Q_2, is $x = c$, then $(T_1 = \xi_1$, $T_2 = \sigma_{X=c}(T_1))$ is a plan for Q. If $\bar{x}_1$ and $\bar{x}_2$ are disjoint, then Q has a plan $(T_1 = \xi_1$, T_2 = ξ_2, T_3 = $T_1 \times T_2)$, of size $|\xi_1| + |\xi_2| + 1$. Otherwise, *i.e.*, $\bar{x}_1 \cap \bar{x}_2 \neq \emptyset$, then Q has a plan $(T_1 = \xi_1, T_2 = \xi_2, T_3 = \rho(T_2), T_4 = T_1 \times T_3, T_5 = \sigma_{X_1 \cap X_2 = \rho(X_1 \cap X_2)}(T_4), T_6 = \pi_{X_1 \cup X_2}(T_4))$, of size $|\xi_1| + |\xi_2| + 4$. Here ρ only renames attributes in $X_1 \cap X_2$.

(b) When Q_1 has a bounded plan ξ_1, and if Q_2 is a projection of a relation atom covered by a constraint $R(X_1 \to U, N)$ in $\mathcal{A}$, then $Q(\bar{z})$ also has a bounded plan as long as $Q_1(\bar{x}_1)$ has bounded output. In this case, a plan for Q_2 is $(T = \xi_1, \mathsf{fetch}(X_1 \in T, R, U))$, which is of size $|\xi_1| + 1$. That is, we instantiate the X_1 attributes of R with (bounded) output of $Q_1(\bar{x}_1)$. This is a typical case in a rewritten query, when we fetch data from D by making use of values retrieved from a view. To check whether Q_1 has bounded output, we expand Q_s with a conjunct $Q_1(\bar{x}_1)$, and inspect $\mathsf{cov}(Q_s(\bar{x}) \wedge Q_1(\bar{x}_1), Q_\varepsilon)$.

We expand Q_s with $Q_1(\bar{x}_1)$ only if $Q_1(\bar{x}_1)$ has a bounded plan using $\mathcal{V}$. It is easy to show that this expansion policy assures that Q_s has a bounded plan since we start with a tautology query $Q_s = Q_\varepsilon$. Hence when we inductively define $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z}))$, it suffices to check whether $Q_s(\bar{x})$ has bounded output only.

(c) When none of these conditions is satisfied, Q is not guaranteed to have a bounded plan.

(8) $Q(\bar{z})$ is $Q_1(\bar{z}) \wedge \neg Q_2(\bar{z})$. For $i \in [1,2]$, let $\mu_i = \mathsf{cov}(Q_s(\bar{x}), Q_i(\bar{z}))$ and $s_i = \mathsf{size}(Q_s(\bar{x}), Q_i(\bar{z}))$. Then

(a) if both μ_1 and μ_2 are true, then $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z}))$ = true, and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = s_1 + s_2 + 1$; otherwise

(b) if μ_1 = true, $Q_2(\bar{z})$ is of the form $\exists \bar{w} R(\bar{x}_1, \bar{x}_2, \bar{w})$, constraint $R(X_1 \to U, N)$ is in $\mathcal{A}$ with $X_1 \cup X_2 = X_1 \cup U = Z$, and if $\mathsf{cov}(Q_s(\bar{x}) \wedge \exists(\bar{z} \setminus \bar{x}_1) Q_1(\bar{z}), Q_\varepsilon)$ = true, then $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z}))$ = true, $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = 2s_1 + 3$; otherwise

(c) $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z}))$ = false, $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = +\infty$.

Note that in case (b), we extend $Q_s(\bar{x})$ to $\mathsf{cov}(Q_s(\bar{x}) \wedge \exists(\bar{z} \setminus \bar{x}_1) Q_1(\bar{z}), Q_\varepsilon)$ = true, for the same reason as for case (7b) above. Moreover, observe the following.

(a) If Q_1 and Q_2 have bounded plans ξ_1 and ξ_2, respectively, then $Q(\bar{z})$ has a bounded plan $(T_1 = \xi_1, T_2 = \xi_2, T_1 - T_2)$ of size at most $|\xi_1| + |\xi_2| + 1$.

(b) Suppose that Q_1 has a bounded plan ξ_1, and Q_2 is the projection of relation atom $R(X_1, X_2, W)$ on $Z = X_1 \cup X_2$. Under constraint $R(X_1 \to U, N)$ in $\mathcal{A}$, we want to make Q_2 bounded, *i.e.,* make $\mathsf{fetch}(X_1 \in T_2, R, U)$ conform to $\mathcal{A}$, where $T_2 = \pi_{X_1}(T_1)$ and $T_1 = \xi_1$. Hence we check whether $\pi_{X_1}(\xi_1)$ has bounded output by inspecting $\mathsf{cov}(Q_s(\bar{x}) \wedge \exists(\bar{z} \setminus \bar{x}_1) Q_1(\bar{z}), Q_\varepsilon)$, to ensure that $\pi_{X_1}(\xi_1)$ computes a set T_2 of a bounded size. If so, then $Q(\bar{z})$ has a bounded plan $(T_1 = \xi_1, T_2 = \pi_{X_1}(T_1), T_3 = \mathsf{fetch}(X \in T_2, R, U), T_4 = \xi_1, T_5 = T_4 - T_3)$, of size $2s_1 + 3$. Here ξ_1 is used twice to ensure that the plan makes a query tree. Note that when Q_s is expanded with $\exists(\bar{z} \setminus \bar{x}_1) Q_1(\bar{z})$, the latter has a bounded plan, as μ_1 = true, and $\exists(\bar{z} \setminus \bar{x}_1) Q_1(\bar{z})$ is expressed by $\pi_{X_1}(T_1)$.

Example 26: We illustrate how $\mathsf{cov}(Q_\varepsilon, Q(x))$ distinguishes different uses of views by means of Q_s, namely, whether to check bounded output of the views.

(a) Recall query Q_2, view V_2 and access schema $\mathcal{A}_1$ from Example 25. We show

how Q_2 is determined to have a bounded plan iff V_2 has bounded output under $\mathcal{A}_1$. The sub-queries of Q_2 are listed as follows:

$$
\begin{aligned}
&Q_2 = \exists x_p, c_4(q_1(x_p, \mathsf{mid}, c_4)),\\
&q_1 = q_2(x_p, \mathsf{mid}, c_4) \wedge c_4 = \text{“movie”},\\
&q_2 = q_3(x_p, \mathsf{mid}) \wedge \mathsf{like}(x_p, \mathsf{mid}, c_4),\\
&q_3 = V_2(x_p) \wedge q_4(\mathsf{mid}),\\
&q_4 = \exists c_3(q_5(\mathsf{mid}, c_3)),\\
&q_5 = q_6(\mathsf{mid}, c_3) \wedge c_3 = 5,\\
&q_6 = q_7(\mathsf{mid}) \wedge \mathsf{rating}(\mathsf{mid}, c_3),\\
&q_7 = \exists c_1, c_2(q_8(c_1, c_2, \mathsf{mid})),\\
&q_8 = q_9(c_1, c_2) \wedge q_{10}(c_1, c_2, \mathsf{mid}),\\
&q_9 = (c_1 = \text{“Universal”}) \wedge (c_2 = \text{“2014”}),\\
&q_{10} = \exists y_m(\mathsf{movie}(\mathsf{mid}, y_m, c_1, c_2)).
\end{aligned}
$$

Given these, $\mathsf{cov}(Q_\varepsilon, Q_2)$ is computed as follows.

$$
\begin{aligned}
\mathsf{cov}(Q_\varepsilon, Q_2) &= \mathsf{cov}(Q_\varepsilon, q_2) = \mathsf{cov}(q_3, Q_\varepsilon) \wedge \mathsf{cov}(Q_\varepsilon, q_3),\\
\mathsf{cov}(Q_\varepsilon, q_3) &= \mathsf{cov}(Q_\varepsilon, q_4) = \mathsf{cov}(Q_\varepsilon, q_5) = \mathsf{cov}(Q_\varepsilon, q_6)\\
&= \mathsf{cov}(q_7, Q_\varepsilon) \wedge \mathsf{cov}(Q_\varepsilon, q_7),\\
\mathsf{cov}(Q_\varepsilon, q_7) &= \mathsf{cov}(Q_\varepsilon, q_8) = \mathsf{cov}(Q_\varepsilon, q_9) \wedge \mathsf{cov}(q_9, Q_\varepsilon)
\end{aligned}
$$

We can verify that $\mathsf{cov}(q_3, Q_\varepsilon) = \mathsf{true}$ iff V_2 has bounded output under $\mathcal{A}_1$, and q_7 and q_9 have bounded output (regardless of V_2). In other words, if V_2 has bounded output then $\mathsf{cov}(q_3, Q_\varepsilon)$, $\mathsf{cov}(q_7, Q_\varepsilon)$ and $\mathsf{cov}(q_9, Q_\varepsilon)$ are all true. Observe also that $\mathsf{cov}(q_9, Q_\varepsilon) = \mathsf{true}$. Hence, $\mathsf{cov}(Q_\varepsilon, Q_2) = \mathsf{true}$ iff $V_2(x_p)$ has bounded output. This is consistent with Example 25. One can verify that when V_2 has bounded output, $\mathsf{size}(Q_\varepsilon, Q_2) = 13$.

(b) Now recall $Q(x) = Q_3(x) \wedge V_3(x)$ from Example 25. We have that $\mathsf{cov}(Q_\varepsilon, Q(x)) = \mathsf{cov}(Q_\varepsilon, Q_3(x)) \wedge \mathsf{cov}(Q_\varepsilon, V_3(x)) = \mathsf{cov}(Q_\varepsilon, Q_3(x))$. Hence $Q(x)$ is topped by $(\mathcal{R}_0, V_3, \mathcal{A}_1, M)$ regardless of whether the view V_3 has bounded output or not. That is, if a view is not included in Q_s by $\mathsf{cov}(Q_\varepsilon, Q_3(x))$, it can be freely used even when it does not have bounded output. □

Observe that $\mathcal{L}_{\mathsf{BR}}(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ is quite different from the rules for $\bar{x}$-controllability [FGL14] and the covered queries for CQ (Chapter 2) and for RA [CF16], particularly in the use of Q_s to check bounded output of views and the function $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ to ensure the bounded size of query plans.

Proof of Theorem 49. We verify the three conditions of effective syntax one by one as follows.

Condition (c) is verified as follows. By the definition of $\mathcal{L}_{\mathsf{BR}}(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, if an FO query Q is in $\mathcal{L}_{\mathsf{BR}}(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, then $|Q| \leqslant \mathsf{size}(Q_\varepsilon, Q) \cdot \mathsf{max}_{V \in \mathcal{V}}|V| \leqslant M \cdot \mathsf{max}_{V \in \mathcal{V}}|V|$, where $|Q|$ is the number of nodes in the query syntax tree of Q. Thus, a PTIME algorithm for checking whether $Q \in \mathcal{L}_{\mathsf{BR}}(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ works as follows: (1) check whether $|Q| \leqslant M \cdot \mathsf{max}_{V \in \mathcal{V}}|V|$; if not, return "no"; otherwise (2) check whether $\mathsf{size}(Q_\varepsilon, Q) \leqslant M$ and $\mathsf{cov}(Q_\varepsilon, Q) = \mathsf{true}$; if so, return "yes"; otherwise return "no". The algorithm is obviously in PTIME with oracle $O_{\mathcal{A}}$ for output boundedness checking.

The other two conditions require more work to validate.

(a) Each FO query Q with an M-bounded rewriting is $\mathcal{A}$-equivalent to a query topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. By definition, an FO query Q with an M-bounded rewriting is $\mathcal{A}$-equivalent to an M-bounded query plan $\xi(Q, \mathcal{V}, \mathcal{R})$ under $\mathcal{A}$. Hence, it suffices to show that for each M-bounded query plan ξ using $\mathcal{V}$ under $\mathcal{A}$, there exists a query Q_ξ topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ such that $\xi \equiv_{\mathcal{A}} Q_\xi$.

We show this by induction on M. More specifically, our induction hypothesis is that for any M-bounded query plan ξ using $\mathcal{V}$ under $\mathcal{A}$, there exists a query Q_ξ topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ such that $Q_\xi \equiv_{\mathcal{A}} \xi$.

Base case. We first show that the induction hypothesis holds when M = 1. In this case, ξ can only be one of the following three forms (see the definition of query plans in Section 5.1): (i) a constant $\{c\}$; (ii) a view predicate $V(\bar{x})$; or (iii) a fetch operator $\mathsf{fetch}(\emptyset, R, X)$ with access constraint $R(\emptyset \to X, N) \in \mathcal{A}$. Define Q_ξ as $x = c$, $V(\bar{x})$ or $\exists \bar{y} R(\bar{y}, \bar{x})$, respectively. Clearly, $Q_\xi \equiv_{\mathcal{A}} \xi$; so it remains to verify whether Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, *i.e.,* whether $\mathsf{cov}(Q_\varepsilon, Q_\xi)$ is true and $\mathsf{size}(Q_\varepsilon, Q_\xi) = 1$. This is a direct consequence of the definition of these two functions. Indeed, case (i) corresponds to case (2) with $\bar{z} = x$; case (ii) corresponds to case (3) with $\bar{z} = \bar{x}$; and case (iii) corresponds to case (5a) with $\bar{z} = \bar{x}$ and $\bar{z}' = \bar{y}$. Hence, Q_ξ is indeed topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$.

Induction step. Suppose that the induction hypothesis holds for $(M-1)$-bounded query plans ξ using $\mathcal{V}$ under $\mathcal{A}$. We next show that the hypothesis also holds for M-bounded query plans ξ. By analyzing the structure of ξ we can distinguish the following cases: (i) $\xi = (\xi', \sigma_{X=c}(\xi'))$ (resp. $(\xi', \sigma_{X=Y}(\xi'))$); (ii) $\xi = (\xi', \pi_Y(\xi'))$; (iii) $\xi = (\xi_1, \xi_2, \xi_1 \times \xi_2)$; (iv) $\xi = (\xi_1, \xi_2, \xi_1 \cup \xi_2)$; (v) $\xi = (\xi_1, \xi_2, \xi_1 - \xi_2)$; or (vi) $\xi = (T = \xi', \mathsf{fetch}(X \in T, R, Y))$. We next show that there exists Q_ξ topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ and $Q_\xi \equiv_{\mathcal{A}} \xi$, by considering each of these cases.

Case (i). We prove the case when $\xi = (\xi', \sigma_{X=c}(\xi'))$; the case when $\xi = (\xi', \sigma_{X=Y}(\xi'))$ is similar. Clearly, ξ' is an $(M-1)$-bounded query plan under $\mathcal{V}$ using $\mathcal{A}$. Hence, by the induction hypothesis, there exists $Q_{\xi'}$ topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M-1)$ such that $Q_{\xi'} \equiv_{\mathcal{A}} \xi'$. Let Q_ξ be $Q_{\xi'} \wedge (x = c)$. Since $Q_{\xi'} \equiv_{\mathcal{A}} \xi'$, we also have that $Q_\xi \equiv_{\mathcal{A}} \xi$. We next show that Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. To see this, consider the conjunction case (7). By the induction hypothesis, $\mathsf{cov}(Q_\varepsilon, Q_{\xi'}) = \mathsf{true}$ and from the base case we know that also $\mathsf{cov}(Q_\varepsilon, x = c) = \mathsf{true}$. Hence, case (7b) applies and $\mathsf{cov}(Q_\varepsilon, Q_\xi) = \mathsf{true}$. With regards to $\mathsf{size}(Q_\varepsilon, Q_\xi)$, observe that $\mathsf{size}(Q_\varepsilon, Q_{\xi'} \wedge x = c)$ is defined in (7b) as $\mathsf{size}(Q_\varepsilon, Q_{\xi'}) + \mathsf{size}(Q_\varepsilon, x = c)$. We know by the induction hypothesis that this is bounded by $(M-1)+1 = M$. Hence, Q_ξ is indeed topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$.

Case (ii). When $\xi = (\xi', \pi_Y(\xi'))$, ξ' is an $(M-1)$-bounded query plan under $\mathcal{V}$ using $\mathcal{A}$. Hence, by the induction hypothesis, there exists $Q_{\xi'}(\bar{z})$ topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M-1)$ such that $\xi' \equiv_{\mathcal{A}} Q_{\xi'}$. Let Q_ξ be $\exists(\bar{z} \setminus \bar{y})\, Q_{\xi'}(\bar{z})$. From $Q_{\xi'} \equiv_{\mathcal{A}} \xi'$ it follows that $Q_\xi \equiv_{\mathcal{A}} \xi$ also holds. We next verify that Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. By projection case (5), there are two cases (5a) and (5b). We next consider these two cases. (5a) If $Q_{\xi'}$ is a relation atom and there is $R(\emptyset \to Y, N)$ in $\mathcal{A}$, then $\mathsf{cov}(Q_\varepsilon, Q_\xi) = \mathsf{true}$, and $\mathsf{size}(Q_\varepsilon, Q_\xi) = 1 \leqslant M$. Thus Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. (5b) By the induction hypothesis, $\mathsf{cov}(Q_\varepsilon, Q_\xi) = \mathsf{cov}(Q_\varepsilon, Q_{\xi'}) = \mathsf{true}$, $\mathsf{size}(Q_\varepsilon, Q_\xi) = \mathsf{size}(Q_\varepsilon, Q_{\xi'}) + 1$. As $Q_{\xi'}$ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M-1)$, $\mathsf{size}(Q_\varepsilon, Q_\xi) \leqslant M - 1 + 1 = M$. Thus, Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$.

Case (iii). When $\xi = (\xi_1, \xi_2, \xi_1 \times \xi_2)$, then ξ_1 is an M_1-bounded query plan and ξ_2 is an M_2-bounded query plan such that $M_1 + M_2 \leqslant M - 1$. Let $Q_{\xi_1}(\bar{x}_1) \equiv_{\mathcal{A}} \xi_1$ and $Q_{\xi_2}(\bar{x}_2) \equiv_{\mathcal{A}} \xi_2$ be the queries topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M_1)$ and $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M_2)$, respectively. Note that $\bar{x}_1 \cap \bar{x}_2 = \emptyset$. Consider $Q_\xi = Q_{\xi_1}(\bar{x}_1) \wedge Q_{\xi_2}(\bar{x}_2)$. Clearly, $Q_\xi \equiv_{\mathcal{A}} \xi$. We show that Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. Since $\mathsf{cov}(Q_\varepsilon, Q_{\xi_1})$ and $\mathsf{cov}(Q_\varepsilon, Q_{\xi_2})$ are true, we know from the conjunction case (7a) that $\mathsf{cov}(Q_\varepsilon, Q_\xi) = \mathsf{true}$ as well. Furthermore, since $\bar{x}_1 \cap \bar{x}_2 = \emptyset$, $\mathsf{size}(Q_\varepsilon, Q_\xi)$ is defined in (7a) as $\mathsf{size}(Q_\varepsilon, Q_{\xi_1}) + \mathsf{size}(Q_\varepsilon, Q_{\xi_2}) + 1$, which is bounded by $M_1 + M_2 + 1 \leqslant M$. Hence, Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$.

Case (iv). The case when $\xi = (\xi_1, \xi_2, \xi_1 \cup \xi_2)$ is dealt with in the same way as the previous case, by using the disjunction case (6).

Case (v). When $\xi = (\xi_1, \xi_2, \xi_1 \setminus \xi_2)$, the case is dealt with in the same way as case (iv), by using the negation case (8).

Case (vi). When $\xi = (S = \xi', \mathsf{fetch}(X \in S, R, Z))$, since ξ is an M-bounded query plan using $\mathcal{V}$ under $\mathcal{A}$, we know that there exists $R(X \to Z', N)$ in $\mathcal{A}$ with $Z \subseteq X \cup Z'$. As before, ξ' is an $(M-1)$-bounded query plan using $\mathcal{V}$ under $\mathcal{A}$. Hence, the induction hypothesis applies. Let $Q_{\xi'}(\bar{x})$ be a query topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M-1)$ such that $Q_{\xi'}(\bar{x}) \equiv_{\mathcal{A}} \xi'$ and consider $Q_{\xi}(\bar{x}, \bar{z}) = Q_{\xi'}(\bar{x}) \wedge \exists \bar{u} R(\bar{x}, \bar{z}, \bar{u})$. Clearly, $Q_{\xi}(\bar{x}, \bar{z}) \equiv_{\mathcal{A}} \xi$. We next verify that Q_{ξ} is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. This follows from the conjunction case (7b). Indeed, by the induction hypothesis we have that $\mathsf{cov}(Q_{\varepsilon}, Q_{\xi'}) = \mathsf{true}$. Furthermore, $\mathsf{cov}(Q_{\varepsilon}, Q_{\xi})$ is defined in (7b) as $\mathsf{cov}(Q_{\varepsilon}(\bar{x}) \wedge Q_1(\bar{x}_1), Q_{\varepsilon})$. Thus $\mathsf{cov}(Q_1(\bar{x}_1), Q_{\varepsilon}) = \mathsf{true}$ provided that $Q_1(\bar{x}_1)$ has bounded output. This follows from the fact that ξ' conforms to $\mathcal{A}$. Hence, $\mathsf{cov}(Q_{\varepsilon}, Q_{\xi}) = \mathsf{true}$. Furthermore, in case (7b) $\mathsf{size}(Q_{\varepsilon}, Q_{\xi})$ is defined as $\mathsf{size}(Q_{\varepsilon}, Q_{\xi'}) + 1$, which is bounded by $(M-1) + 1 = M$. In other words, Q_{ξ} is also topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$.

(b) Every query topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ has an M-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$. We show that every query Q topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ has an $\mathsf{size}(Q_{\varepsilon}, Q)$-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$. Since $\mathsf{size}(Q_{\varepsilon}, Q) \leqslant M$ for topped queries, Q has indeed an M-bounded rewriting by the definition of $\mathcal{L}_{\mathsf{BR}}(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. The proof is by induction on the structure of Q. In the following, when $\bar{x}$ is tuple of variable we denote by X a corresponding set of attributes, and vice versa.

Base case. We first show that the hypothesis holds when Q is either (b1) a relation $R(\bar{z})$; (b2) $z = c$; or (b3) a view $V(\bar{z})$. For case (b1), consider case (1). Since Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, this implies that $\mathsf{cov}(Q_{\varepsilon}, R(\bar{z})) = \mathsf{true}$. Note, however, that only case (1a) applies. Indeed, to apply (1b) $Q_s(\bar{x})$ is required to have at least one free variable. By contrast, in the base case Q_s is the tautology query Q_{ε}. From case (1a) we know that $\mathsf{cov}(Q_{\varepsilon}, R(\bar{z})) = \mathsf{true}$ and $\mathsf{size}(Q_{\varepsilon}, R(\bar{z})) = 1$. Hence, if Q is topped, it is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, 1)$. To see that Q has indeed an 1-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$, observe that case (1a) implies that there exists an access constraint $R(\emptyset \to Z, N)$ in $\mathcal{A}$. Given this, $\xi_Q = \mathsf{fetch}(\emptyset, R, Z)$ is the required 1-bounded query plan.

For cases (b2) and (b3), cases (2) and (3), respectively, imply that when Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ it is in fact topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, 1)$. It now suffices to show that in both these cases Q has an 1-bounded query plan using $\mathcal{V}$ under $\mathcal{A}$. Clearly, $\xi_Q = \{c\}$ and $\xi_Q(Z) = V(Z)$ are such query plans, respectively.

Hence, when Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, 1)$ then Q has an 1-bounded query plan using $\mathcal{V}$ under $\mathcal{A}$.

Induction step. Consider the following cases on the structure of $Q(\bar{z})$. The cases below correspond to the different cases presented. We number them accordingly in the proof below.

(4) $Q(\bar{z})$ is $Q'(\bar{z}) \wedge (x' = y')$. For $Q(\bar{z})$ to be topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, $\mathsf{cov}(Q_{\varepsilon}, Q(\bar{z}))$

must be true. As defined in the syntax, this happens when $\mathsf{cov}(Q_\varepsilon, Q'(\bar{z})) =$ true. Furthermore, $\mathsf{size}(Q_\varepsilon, Q(\bar{z})) = \mathsf{size}(Q_\varepsilon, Q'(\bar{z})) + 1$. Hence, $Q'(\bar{z})$ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M-1)$. In other words, the induction hypothesis applies to $Q'(\bar{z})$ and there exists a $\mathsf{size}(Q_\varepsilon, Q'(\bar{z}))$-bounded query plan $\xi_{Q'}$ of $Q'(\bar{z})$. Consider $\xi_Q = (T_1 = \xi_{Q'}, T_2 = \sigma_{X'=Y'}(T_1))$. Clearly, this is a $\mathsf{size}(Q_\varepsilon, Q(\bar{z}))$-bounded query plan of Q. Note that $\mathsf{size}(Q_\varepsilon, Q(\bar{z})) \leqslant (M-1)+1 = M$.

(5) $Q(\bar{z})$ is $\exists z'\, Q'(\bar{z}', \bar{z})$. For $Q(\bar{z})$ to be topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, $\mathsf{cov}(Q_\varepsilon, Q(\bar{z}))$ must be true. Case (5) distinguishes between the following two cases: (5a) $Q'(\bar{z}', \bar{z})$ is a relation atom $R(\bar{z}', \bar{z})$ and there exists an access constraint $R(\emptyset \to Z, N)$ in $\mathcal{A}$; and (5b) $Q'(\bar{z}', \bar{z})$ is such that $\mathsf{cov}(Q_\varepsilon, Q') =$ true. Furthermore, in case (5a) $\mathsf{size}(Q_\varepsilon, Q') = 1$. We cannot use the induction hypothesis here. However, it is easy to see that $\xi_Q = \mathsf{fetch}(\emptyset, R, Z)$ is an 1-bounded query plan of Q. For case (5b), $\mathsf{size}(Q_\varepsilon, Q) = \mathsf{size}(Q_\varepsilon, Q') + 1$. Hence, $\mathsf{size}(Q_\varepsilon, Q') \leqslant M-1$ and the induction hypothesis applies to $Q(\bar{z}', \bar{z})$. In other words, there exists a $\mathsf{size}(Q_\varepsilon, Q')$-bounded query plan $\xi_{Q'}$ of Q'. Consider $\xi_Q = (T_1 = \xi_{Q'}, T_2 = \pi_Z(T_1))$. Clearly this is a $\mathsf{size}(Q_\varepsilon, Q)$-bounded query plan for Q. Note that $\mathsf{size}(Q_\varepsilon, Q) \leqslant (M-1)+1 = M$.

(6) $Q(\bar{z})$ is $Q_1(\bar{z}) \vee Q_2(\bar{z})$. For $Q(\bar{z})$ to be topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, $\mathsf{cov}(Q_\varepsilon, Q(\bar{z}))$ must be true. This implies, by (6a), that both $\mathsf{cov}(Q_\varepsilon, Q_1(\bar{z}))$ and $\mathsf{cov}(Q_\varepsilon, Q_2(\bar{z}))$ must be true. Furthermore, in this case $\mathsf{size}(Q_\varepsilon, Q(\bar{z})) = s_1 + s_2 + 1$ where $s_i = \mathsf{size}(Q_\varepsilon, Q_i(\bar{z}))$. Hence, $Q_1(\bar{z})$ and $Q_2(\bar{z})$ are topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, N_1)$ and $(\mathcal{R}, \mathcal{V}, \mathcal{A}, N_2)$, respectively, for some $N_1 + N_2 \leqslant M-1$. As before, the induction hypothesis applies to $Q_1(\bar{z})$ and $Q_2(\bar{z})$. Let ξ_{Q_1} and ξ_{Q_2} be the s_1 and s_2-bounded query plans for $Q_1(\bar{z})$ and $Q_2(\bar{z})$, respectively. Consider $\xi_Q = (T_1 = \xi_{Q_1}, T_2 = \xi_{Q_2}, T_1 \cup T_2)$. Clearly, this is an (s_1+s_2+1)-bounded query plan for $Q(\bar{z})$. Recall that $\mathsf{size}(Q_\varepsilon, Q(\bar{z})) = s_1 + s_2 + 1$ and note that $s_1 + s_2 + 1 \leqslant N_1 + N_2 + 1 \leqslant (M-1)+1$.

(7) $Q(\bar{z})$ is $Q_1(\bar{x}_1) \wedge Q_2(\bar{x}_2)$. For $Q(\bar{z})$ to be topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, $\mathsf{cov}(Q_\varepsilon, Q(\bar{z}))$ must be true. Case (7) in distinguishes between the following two cases: (7a) both $\mathsf{cov}(Q_\varepsilon, Q_1(\bar{x}_1))$ and $\mathsf{cov}(Q_\varepsilon, Q_2(\bar{x}_2))$ are true; and (7b) $\mathsf{cov}(Q_\varepsilon, Q_1(\bar{x}_1)) =$ true and Q_2 is a projection of a relation atom $\exists \bar{w}\, R(\bar{x}_1, \bar{x}_2', \bar{w})$ such that $\bar{x}_2 = \bar{x}_1 \cup \bar{x}_2'$.

Case (7a) can be verified in a similar way as case (6a). More specifically, the induction hypothesis can be applied to $Q_1(\bar{x}_1)$ and $Q_2(\bar{x}_2)$. Let ξ_{Q_i} be a $\mathsf{size}(Q_\varepsilon, Q_i(\bar{x}_i))$-bounded query plan for $Q_i(\bar{x}_i)$, for $i = 1,2$. Let $s_i = \mathsf{size}(Q_\varepsilon, Q_i(\bar{x}_i))$ for $i = 1,2$. As defined in case (7a), $\mathsf{size}(Q_\varepsilon, Q(\bar{z})) = s_1 + s_2 + \lambda$, where λ is a natural number that depends on the type of queries in the conjunction. We show that $Q(\bar{z})$ has a $\mathsf{size}(Q_\varepsilon, Q(\bar{z}))$-bounded query plan by combining ξ_{Q_1} and ξ_{Q_2} into a query plan ξ_Q for Q. More specifically, when, *e.g.*, Q_1, is of form $(x = c)$, $\lambda = 0$ and $\xi_Q = (T_1 = \xi_1, T_2 = \sigma_{X=c}(T_1))$ is indeed (s_1+s_2)-bounded query plan for Q using $\mathcal{V}$ under $\mathcal{A}$. When $\bar{x}_1 \cap \bar{x}_2 = \emptyset$, $\lambda = 1$ and $\xi_Q = (T_1 = \xi_{Q_1}, T_2 = \xi_{Q_2}, T_3 = T_1 \times T_2)$ is indeed an (s_1+s_2+1)-bounded query plan for Q using $\mathcal{V}$ under $\mathcal{A}$. Finally, when $\bar{x}_1 \cap \bar{x}_2 \neq \emptyset$, $\lambda = 4$ and $\xi_Q = (T_1 = \xi_{Q_1}, T_2 = \xi_{Q_2}$, $T_3 = \rho(T_2)$, $T_4 = T_1 \times T_3$, $T_5 = \sigma_{X_1 \cap X_2 = \rho(X_1 \cap X_2)}(T_4)$, $T_6 = \pi_{X_1 \cup X_2}(T_5))$ is an (s_1+s_2+4)-bounded query plan for Q using $\mathcal{V}$ under $\mathcal{A}$. Here, ρ only renames attributes in $X_1 \cap X_2$.

For case (7b), observe that $\mathsf{cov}(Q_\varepsilon, Q(\bar{z})) = \mathsf{true}$ if additionally $\mathsf{cov}(Q_\varepsilon(\bar{x}) \wedge Q_1(\bar{x}_1), Q_\varepsilon) = \mathsf{true}$. This condition simply requires Q_1 to have bounded output. Furthermore, there must exist an access constraint $R(X_1 \to U, N)$ in $\mathcal{A}$ with $X_1 \cup X_2' = X_1 \cup U = X_2$. We also know that $\mathsf{size}(Q_\varepsilon, Q(\bar{z})) = \mathsf{size}(Q_\varepsilon, Q_1(\bar{x}_1)) + 1$. Hence, $Q_1(\bar{x}_1)$ is in fact topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M-1)$ and thus the induction hypothesis applies to $Q_1(\bar{x}_1)$. Let ξ_{Q_1} be the $\mathsf{size}(Q_\varepsilon, Q_1)$-bounded query plan for Q_1 using $\mathcal{V}$ under $\mathcal{A}$. Consider $\xi_Q = (T_1 = \xi_{Q_1}, T_2 = \mathsf{fetch}(X_1 \in T_1, R, U))$. This is clearly an (s_1+1)-bounded query plan for Q. It conforms to $\mathcal{A}$ because Q_1 has bounded output. Note that $\mathsf{size}(Q_\varepsilon, Q) = s_1 + 1 \leqslant (M-1)+1 = M$. Hence, the induction hypothesis also holds for $Q(\bar{z})$.

(8) $Q(\bar{z})$ is $Q_1(\bar{z}) \wedge \neg Q_2(\bar{z})$. For $Q(\bar{z})$ to be topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, $\mathsf{cov}(Q_\varepsilon, Q(\bar{z}))$ must be true. Case (8) distinguishes between the following two cases: (8a) $\mathsf{cov}(Q_\varepsilon, Q_i(\bar{z})) = \mathsf{true}$ for both $i = 1$ and $i = 2$; and (8b) $\mathsf{cov}(Q_\varepsilon, Q_1(\bar{z})) = \mathsf{true}$, and $Q_2(\bar{z})$ is of the form $\exists \bar{w} R(\bar{x}_1, \bar{x}_2, \bar{w})$ with $\bar{z} = \bar{x}_1 \cup \bar{x}_2$ and $R(X_1 \to U, N) \in \mathcal{A}$ with $X_1 \cup X_2 = X_1 \cup U = Z$.

Here, case (8a) can be verified in a similar way as case (6a) and we omit the details of this verification. For case (8b), we first want to verify that $Q_1(\bar{z})$ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M-1)$. We already know that $\mathsf{cov}(Q_\varepsilon, Q_1(\bar{z})) = \mathsf{true}$ and since $\mathsf{size}(Q_\varepsilon, Q(\bar{z})) = 2 \cdot \mathsf{size}(Q_\varepsilon, Q_1(\bar{z})) + 3$, it follows that $\mathsf{size}(Q_\varepsilon, Q_1(\bar{z})) \leqslant \lfloor (M-3)/2 \rfloor$. Hence the induction hypothesis applies and there exists a $\mathsf{size}(Q_\varepsilon, Q_1(\bar{z}))$-bounded query plan ξ_{Q_1} of Q_1. Case 8(b) also requires $\mathsf{cov}(Q_\varepsilon \wedge \exists(\bar{z} \setminus \bar{x}_1) Q_1(\bar{z}), Q_\varepsilon) = \mathsf{true}$. Thus $\mathsf{cov}(\exists(\bar{z} \setminus \bar{x}_1) Q_1(\bar{z}), Q_\varepsilon) = \mathsf{true}$. This implies that $\exists(\bar{z} \setminus \bar{x}_1) Q_1(\bar{z})$ has bounded output. Consider $\xi_Q = (T_1 = \xi_{Q_1}, T_2 = \pi_{X_1}(T_1), T_3 = \mathsf{fetch}(X \in T_2, R, U), T_4 = \xi_{Q_1}, T_5 = T_4 - T_3)$, written in this way to ensure that the plan forms a query tree. It is readily verified that this is a query plan for $Q(\bar{z})$ that conforms to $\mathcal{A}$. Indeed, this query plan evaluates the equivalent query $Q_1(\bar{z}) \wedge \neg(Q_1(\bar{z}) \wedge Q_2(\bar{z}))$. Furthermore, note that $\pi_{X_1}(\xi_{Q_1})$ is of bounded output and thus ξ_Q conforms to $\mathcal{A}$. Hence, ξ_Q is a $\mathsf{size}(Q_\varepsilon, Q)$-bounded query plan. Note that $\mathsf{size}(Q_\varepsilon, Q) \leqslant 2(\lfloor (M-3)/2 \rfloor + 3 \leqslant M$. □

Remark. Function $\mathsf{cov}(Q_\varepsilon, Q(\bar{z}))$ helps us distinguish views that need to have bounded output from those that do not have to when we rewrite queries using views. When $\mathcal{R}, \mathcal{V}, \mathcal{A}$ or M is not necessarily fixed, it can still help us identify bounded rewriting as long as a set of syntactic rules or an effective procedure is in place for us to check whether certain views have bounded output. We expect to find such rules or procedure for simple views commonly used in practice, *e.g.,* views in CQ. At the very least, the function can provide us with a heuristic algorithm to make practical use of bounded rewriting. Function $\mathsf{size}(Q_\varepsilon, Q(\bar{z}))$ helps us ensure that our query rewriting plans do not exceed a given bound.

5.3.2 An Effective Syntax for Output Boundedness

We next develop an effective syntax for FO queries that have bounded output under an access schema $\mathcal{A}$, as a realization of the oracle used in topped queries of Section 5.3.1.

Given a relational schema $\mathcal{R}$ and access schema $\mathcal{A}$ over $\mathcal{R}$, we identify a class $\mathcal{L}_{BO}(\mathcal{R}, \mathcal{A})$ of FO queries, referred to as *size-bounded queries* under $\mathcal{A}$. We show that it serves as an linear-time effective syntax for FO queries with bounded output under $\mathcal{A}$.

Theorem 50: *For any access schema $\mathcal{A}$ over relational schema $\mathcal{R}$,*

(a) each FO *query that has bounded output under $\mathcal{A}$ is $\mathcal{A}$-equivalent to a query in $\mathcal{L}_{BO}(\mathcal{R}, \mathcal{A})$;*

(b) each FO *query in $\mathcal{L}_{BO}(\mathcal{R}, \mathcal{A})$ has bounded output under $\mathcal{A}$; and*

(c) it takes $O(|Q|)$ time to check whether an FO *query Q is in $\mathcal{L}_{BO}(\mathcal{R}, \mathcal{A})$.* □

We next define the class $\mathcal{L}_{BO}(\mathcal{R}, \mathcal{A})$. The idea is to enumerate distinct answers of size-bounded queries over $\mathcal{R}$.

Class $\mathcal{L}_{BO}(\mathcal{R}, \mathcal{A})$. An FO query $Q_o(\bar{x})$ is in $\mathcal{L}_{BO}(\mathcal{R}, \mathcal{A})$ if it is of the following form:

$$Q(\bar{x}) = Q'(\bar{x}) \wedge \exists \bar{x}_1, \ldots, \bar{x}_{K+1} \left((Q'(\bar{x}_1) \wedge \cdots \wedge Q'(\bar{x}_{K+1})) \rightarrow (\bigvee_{i,j \in [1,K+1], i \neq j} x_i = x_j\right),$$

where K is any natural number.

Intuitively, a query Q is in the class $\mathcal{L}_{OB}(\mathcal{R}, \mathcal{A})$ if there exist a number K and an FO query Q' that satisfy the above form. Observe that if $Q \in \mathcal{L}_{OB}(\mathcal{R}, \mathcal{A})$, then for any $D \models \mathcal{A}$, $|Q'(D)| \leqslant K$. This gives a simple proof of Theorem 50, as follows.

Proof: <u>(a) Theorem 50(a)</u>. Since $Q(\bar{x})$ has bounded-output under $\mathcal{A}$, by the definition of bounded output queries, we know that there exists a natural number K such that, for any database $D \models \mathcal{A}$, $|Q(D)| \leqslant K$. Construct $Q'(\bar{x})$ from $Q(\bar{x})$ as follows:

$$Q'(\bar{x}) = Q(\bar{x}) \wedge \exists \bar{x}_1, \ldots, \bar{x}_{K+1}\, ((Q(\bar{x}_1) \wedge \cdots Q(\bar{x}_{K+1})) \to (\bigvee_{i,j \in [1,K+1], i \neq j} (x_i = x_j))).$$

Obviously that $Q'(\bar{x})$ is in $\mathcal{L}_{BO}(\mathcal{R}, \mathcal{A})$. We next show that $Q'(\bar{x}) \equiv_{\mathcal{A}} Q(\bar{x})$. Indeed, since $Q(\bar{x})$ has output bounded by K, the Boolean sub-query $\exists \bar{x}_1, \ldots, \bar{x}_{K+1}\, (Q(\bar{x}_1) \wedge \cdots Q(\bar{x}_{K+1})) \to (\bigvee_{i,j \in [1,K+1], i \neq j} (x_i = x_j))$ of $Q'(\bar{x})$ is always *true*. Thus $Q'(\bar{x}) \equiv_{\mathcal{A}} Q(\bar{x})$.

(b) Theorem 50(b). Consider query $Q(\bar{x}) = Q'(\bar{x}) \wedge \exists \bar{x}_1, \ldots, \bar{x}_{K+1}\, ((Q'(\bar{x}_1) \wedge \cdots \wedge Q'(\bar{x}_{K+1})) \to (\bigvee_{i,j \in [1,K+1], i \neq j} x_i = x_j))$ in $\mathcal{L}_{BO}(\mathcal{R}, \mathcal{A})$. We show that for any $D \models \mathcal{A}$, $|Q(D)| \leqslant K$. Indeed, observe that, for any D, if $Q'(D)$ contains more than K answer tuples, the Boolean sub-query $(Q'(\bar{x}_1) \wedge \cdots \wedge Q'(\bar{x}_{K+1})) \to (\bigvee_{i,j \in [1,K+1], i \neq j} x_i = x_j)$ is *false*. Thus $Q(D) = \emptyset$, *i.e.*, $|Q(D)| \leqslant K$. if $Q'(D)$ contains no more than K answer tuples, then $Q(D) = Q'(D)$. Thus $|Q(D)| \leqslant K$. That is, Q has bounded output.

(b) Theorem 50(c). This is verified by the definition of the class $\mathcal{L}_{BO}(\mathcal{R}, \mathcal{A})$. □

Added functionality to existing DBMS. Capitalizing on the effective syntax, we can develop algorithms (a) to check whether a given FO query Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ in PTIME; and if so, (b) to generate a bounded query plan ξ for Q using $\mathcal{V}$. The existence of these algorithms are warranted by Theorem 49 and Theorem 50. We can then support bounded rewriting using views on top of commercial DBMS as follows. Given an application, a database schema $\mathcal{R}$ and a resource bound M are first determined, based on the application and available resources, respectively. Then, a set $\mathcal{V}$ of views can be selected following [ALK$^+$13], and a set $\mathcal{A}$ of access constraints can be discovered [CF16, CFY14]. After these are in place, given an FO query Q posed on a dataset D that satisfies $\mathcal{A}$, we check whether Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, where we use the syntax $\mathcal{L}_{BO}(\mathcal{R}, \mathcal{A})$ as the oracle $O_{\mathcal{A}}$ for checking queries with bounded output. If so, we generate a bounded query plan ξ for Q using $\mathcal{V}$, by using the algorithms described above. Then we can compute $Q(D)$ by executing ξ with the existing DBMS. Moreover, incremental methods for maintaining the views [ALK$^+$13] and (the indices of) access constraints [CF16] have already been developed, to cope with updates to D. Putting these together, we can expect to efficiently answer a number of FO queries Q in (possibly big) D that have a bounded rewriting.

Summary

We have formalized bounded query rewriting using views under access constraints, studied the bounded rewriting problem $\mathsf{VBRP}(\mathcal{L})$ for CQ, UCQ, $\exists\mathsf{FO}^+$and FO, and established their upper and lower bounds, all matching, when M, $\mathcal{R}$ and $\mathcal{A}$ are fixed or not. We have also provided effective syntax for FO queries with a bounded rewriting and for FO queries with bounded output.

Part III

Boundedly Evaluable Graph Queries

Chapter 6

Bounded Evaluability on Graphs

This chapter extends bounded evaluability from relational queries to graph queries. Like relational queries, it is cost-prohibitive to find matches $Q(G)$ of a pattern query Q in a big graph G. We approach this by fetching a small subgraph G_Q of G such that $Q(G_Q) = Q(G)$. We show that many practical patterns are boundedly evaluable under access constraints $\mathcal{A}$ commonly found in real life, such that G_Q can be identified in time determined by Q and $\mathcal{A}$ only, independent of the size $|G|$ of G. This holds no matter whether pattern queries are localized (*e.g.,* via subgraph isomorphism) or non-localized (graph simulation). We provide algorithms to decide whether a pattern Q is boundedly evaluable, and if so, to generate a query plan that computes $Q(G)$ by accessing G_Q, in time independent of $|G|$. When Q is not boundedly evaluable, we give an algorithm to extend access constraints and make Q bounded in G. Using real-life data, we experimentally verify the effectiveness of the approach, *e.g.,* about 60% of queries are boundedly evaluable for subgraph isomorphism, and for such queries our approach outperforms the conventional methods by 4 orders of magnitude.

Given a pattern query Q and a graph G, *graph pattern matching* is to find the set $Q(G)$ of matches of Q in G. It is used in, *e.g.,* social marketing, knowledge discovery, mobile network analysis, intelligence analysis for identifying terrorist organizations [ORK+13], and the study of adolescent drug use [FE13], just to name a few.

When G is big, graph pattern matching is cost-prohibitive. Facebook has 1.26 billion nodes and 140 billion links in its social graph, about 300PB of user data [Smi14]. When the size $|G|$ of G is 1PB, a linear scan of G takes 1.9 days using SSD with scanning speed of 6GB/s. Worse still, graph pattern matching is intractable if it is defined with subgraph isomorphism [Ull76], and it takes $O(|Q||G|)$-time if we use graph simulation [HHK95a], where $|G| = |V| + |E|$ and $|Q| = |V_Q| + |E_Q|$.

Can we still efficiently compute exact answers $Q(G)$ when G is big while we have constrained resources, such as a single processor? We approach this by *making big graphs small*, capitalizing on a set $\mathcal{A}$ of access constraints, which are a combination of indices and simple cardinality constraints defined on the labels of neighboring nodes of G. We determine whether Q is *boundedly evaluable* under $\mathcal{A}$, *i.e.,* for *all* graphs G that satisfy $\mathcal{A}$, there exists a subgraph $G_Q \subset G$ such that

(a) $Q(G_Q) = Q(G)$, and

(b) the size $|G_Q|$ of G_Q and the time for identifying G_Q are both determined by $\mathcal{A}$ and Q only, *independent of* $|G|$.

If Q is boundedly evaluable, we can generate a query plan that for all G satisfying $\mathcal{A}$, computes $Q(G)$ by accessing (visiting and fetching) a small G_Q in time *independent of* $|G|$, no matter how big G is. Otherwise, we will identify extra access constraints on an input G and make Q bounded in G.

A large number of real-life queries are boundedly evaluable under simple access constraints, as illustrated below.

Example 27: Consider IMDbG [IMD], a graph G_0 in which nodes represent movies, casts, and awards from 1880 to 2014, and edges denote various relationships between the nodes. An example search on IMDbG is to *find pairs of first-billed actor and actress (main characters) from the same country who co-stared in a award-winning film released in 2011-2013*.

The search can be represented as a pattern query Q_0 shown in Fig. 6.1. Graph pattern matching here is to find the set $Q_0(G_0)$ of matches, *i.e.,* subgraphs G' of G_0 that are isomorphic to Q_0; we then extract and return actor-actress pairs from each match G'. The challenge is that G_0 is large: the IMDbG graph has 5.1 million nodes and 19.5

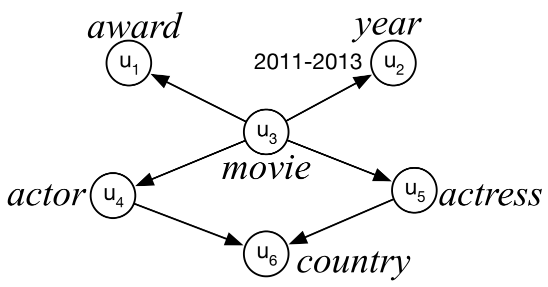


Figure 6.1: Pattern query Q_0 on IMDbG

million edges. Add to this that subgraph isomorphism is NP-complete.

Not all is lost. Using simple aggregate queries one can readily find the following *real-life* cardinality constraints on the movie dataset from 1880–2014: (1) in each year, every award is presented to no more than 4 movies (C1); (2) each movie has at most 30 first-billed actors and actresses (C2), and each person has only one country of origin (C3); and (3) there are no more than 135 years (C4, *i.e.,* 1880-2014), 24 major movie awards (C5) and 196 countries (C6) in total [IMD]. An index can be built on the labels and nodes of G_0 for each of the constraints, yielding a set $\mathcal{A}_0$ of 8 access constraints.

Under $\mathcal{A}_0$, pattern Q_0 is *boundedly evaluable*. We can find $Q_0(G_0)$ by accessing *at most* 17923 nodes and 35136 edges in G_0, *regardless of* the size of G_0, by the following query plan:

(a) identify a set V_1 of 135 year nodes, 24 award nodes and 196 country nodes, by using the indices for constraints C4-C6;

(b) fetch a set V_2 of at most $24 \times 3 \times 4 = 288$ award-winning movies released in 2011–2013, with no more than $288 \times 2 = 576$ edges connecting movies to awards and years, by using those award and year nodes in V_1 and the index for C1;

(c) fetch a set V_3 of at most $(30+30) * 288 = 17280$ actors and actresses with 17280 edges, using V_2 and the index for C2;

(d) connect the actors and actresses in V_3 to country nodes in V_1, with at most 17280 edges by using the index for C3. Output (actor, actress) pairs connected to the same country in V_1.

The query plan visits at most 135 + 24 + 196 + 288 + 17280 = 17923 nodes, and 576 + 17280 + 17280 = 35136 edges, by using the cardinality constraints and indices in $\mathcal{A}_0$, as opposed to tens of millions of nodes and edges in IMDbG. □

This example tells us that graph pattern matching is feasible in big graphs within constrained resources, by making use of boundedly evaluable pattern queries. To develop a practical approach out of the idea, several questions have to be answered. (1) Given a pattern query Q and a set $\mathcal{A}$ of access constraints, can we determine whether Q is boundedly evaluable under $\mathcal{A}$? (2) If Q is boundedly evaluable, how can we generate a query plan to compute $Q(G)$ in big G by accessing a bounded G_Q? (3) If Q is not bounded, can we make it "bounded" in G by adding simple extra constraints? (4) Does the approach work on both localized queries (*e.g.,* via subgraph isomorphism) and non-localized queries (via graph simulation)?

These questions are not easy to answer. Given that bounded evaluability for relational queries has already been developed in Part I, one may attmpt to handle graph pattern queries by storing data graphs in relations of edges and expressing graph patterns by relational queries over the schemas of edge relations. However, there are a number of issues about this method. First, as shown in Example 27, access constraints in $\mathcal{A}_0$ used by the bounded plan are about cardinality constraints among the labels, which are the data values instead of attributes in the relational representation (*i.e.,* edge relations) of the data graphs. Therefore, access constraints $R(X \to Y, N)$ defined in Chapter 2 for relations cannot express constraints over graph data needed for answering pattern queries. Second, as shown in Chapter 2, bounded analysis for relational queries is quite expensive, *e.g.,*EXPSPACE-hard for CQ queries already, it would be an overkill to directly use bounded evaluability for relational queries in order to answer graph pattern queries, as those relational queries interpreting graph patterns only make a small sub-class of the conventional relational queries, which should have a much lower complexity on bounded evaluability analysis. Third, patterns by graph simulation can express recursion and are non-localized, which are beyond the scope of RA queries we've analyzed so far in Part I. Therefore, it is necessary to develop a native solution to handle graph pattern matching queries over big data graphs.

Overview. This chapter aims to answer these questions and to establish bounded evaluability for graphs. The main results are as follows.

(1) We introduce bounded evaluability for graph pattern queries (Section 6.1). We formulate access constraints on graphs, and define boundedly evaluable pattern queries. We also show how to find simple access constraints from real-life data.

(2) We characterize boundedly evaluable *subgraph queries* Q, *i.e.,* patterns defined by subgraph isomorphism (Section 6.2). We identify a *sufficient and necessary* condition to decide whether Q is boundedly evaluable under a set $\mathcal{A}$ of access constraints. Using the condition, we develop a decision algorithm in $O(|\mathcal{A}||E_Q| + ||\mathcal{A}|||V_Q|^2)$ time, where

$|Q| = |V_Q| + |E_Q|$, and $||\mathcal{A}||$ is the number of constraints in $\mathcal{A}$. The cost is *independent of* big graph G, and query Q is typically small in practice.

(3) We provide an algorithm to generate query plans for boundedly evaluable subgraph queries (Section 6.3). After Q is found boundedly evaluable under $\mathcal{A}$, the algorithm generates a query plan that, given a graph G that satisfies $\mathcal{A}$, accesses a subgraph G_Q of size independent of $|G|$, in $O(|V_Q||E_Q||\mathcal{A}|)$ time. Moreover, we show that the plan is *worst-case-optimal*, *i.e.,* for each input Q and $\mathcal{A}$, the *largest* G_Q it finds from all graphs G that satisfy $\mathcal{A}$ is *minimum* among all worst-case G_Q identified by all other plans.

(4) If Q is not bounded under $\mathcal{A}$, we make it *instance-bounded* (Section 6.4). That is, for a given graph G that satisfies $\mathcal{A}$, we find an extension $\mathcal{A}_M$ of $\mathcal{A}$ such that under $\mathcal{A}_M$, we can find $G_Q \subset G$ in time decided by $\mathcal{A}_M$ and Q, and $Q(G_Q) = Q(G)$. We show that when the size of indices in $\mathcal{A}_M$ is predefined, the problem for deciding the existence of $\mathcal{A}_M$ is in low polynomial time (PTIME), but it is log-APX-hard to find a minimum $\mathcal{A}_M$. When $\mathcal{A}_M$ is unbounded, all query loads can be made instance-bounded by adding simple access constraints.

(5) We extend the study to *simulation queries*, *i.e.,* patterns interpreted by graph simulation (Section 6.5). It is more challenging to cope with the *non-localized* and *recursive* nature of simulation queries. Nonetheless, we provide a characterization of boundedly evaluable simulation queries. We also show that our algorithms for checking bounded evaluability, generating query plans, and for making queries instance-bounded can be adapted to simulation queries, with the same complexity.

(6) We experimentally evaluate our algorithms using real-life data (Section 6.6). We find that our approach is effective for both localized and non-localized queries: (a) on graphs G of billions of nodes and edges [Web], our query plans outperform the conventional methods that computes $Q(G)$ directly by *4 and 3 orders of magnitude* on average, for subgraph and simulation queries, respectively, accessing at most *0.0032%* of the data in G; (b) 60% (resp. 33%) of subgraph (resp. simulation) queries are boundedly evaluable under simple access constraints; and (c) *all queries* can be made instance-bounded in G by extending constraints and accessing 0.016% of extra data in G; and 95% become instance-bounded by accessing at most 0.009% extra data. Our algorithms are efficient: they take at most 37ms to decide whether Q is boundedly evaluable and to generate an optimal query plan for all Q and constraints tested.

This work is the first effort to study boundedly evaluable graph queries, from fundamental problems to practical algorithms. It suggests an approach to querying graphs: (1) given a query Q, we check whether Q is boundedly evaluable under a set $\mathcal{A}$ of access constraints; (2) if so, we generate a query plan that given a graph G satisfying $\mathcal{A}$, computes $Q(G)$ by accessing G_Q of size independent of $|G|$, no matter how big G grows; (3) if not, we make Q instance-bounded in G with extra simple constraints. The approach works for *both* localized subgraph queries *and* non-localized simulation queries.

Given the prohibitive cost of querying big graphs, this approach helps even when only *limited queries* are boundedly evaluable. In fact, we find that many queries on real-life datasets are actually boundedly evaluable under very simple access constraints. Moreover, when a finite set of queries is not boundedly evaluable, we can make them instance-bounded.

6.1 Boundedly Evaluable Graph Pattern Queries

In this section we define access schema on graphs and boundedly evaluable graph pattern queries. We start with a review of graphs and patterns. Assume an alphabet Σ of labels.

Graphs. A data graph is a node-labeled directed graph $G = (V, E, f, \nu)$, where (1) V is a finite set of nodes; (2) $E \subseteq V \times V$ is a set of edges, in which (v, v') denotes the edge from v to v'; (3) $f()$ is a function such that for each node v in V, $f(v)$ is a label in Σ, *e.g.,* year; and (4) $\nu(v)$ is the attribute value of $f(v)$, *e.g.,* year = 2011.

We write G as (V, E) or (V, E, f) when it is clear from the context. The *size* of G, denoted by $|G|$, is defined to be the total number of nodes and edges in G, *i.e.,* $|G| = |V| + |E|$.

Remark. To simplify the discussion, we do not explicitly define edge labels. Nonetheless, our techniques can be readily adapted to edge labels: for each labeled edge e, we can insert a "dummy" node to represent e, carrying e's label.

Labeled set. For a set $S \subseteq \Sigma$ of labels, we say that $V_S \subseteq V$ is a *S-labeled set* of G if (a) $|V_S| = |S|$ and (b) for each label l_S in S, there exists a node v in V_S such that $f(v) = l_S$. In particular, when $S = \emptyset$, the S-labeled set in G is $\emptyset$.

Common neighbors. A node v is called a *neighbor* of another node v' in G if either (v, v') or (v', v) is an edge in G. We say that v is a *common neighbor* of a set V_S of nodes in G if for all nodes v' in V_S, v is a neighbor of v'. In particular, when V_S is $\emptyset$, all nodes of G are common neighbors of V_S.

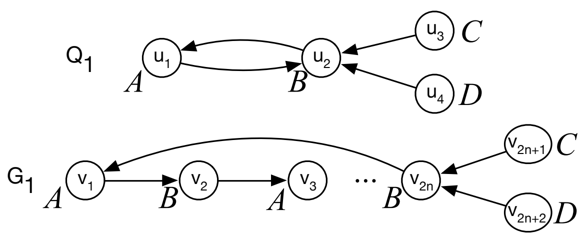


Figure 6.2: Pattern query Q_1 and data graph G_1

Subgraphs. Graph $G_s = (V_s, E_s, f_s, \nu_s)$ is a *subgraph* of G if $V_s \subseteq V$, $E_s \subseteq E$, and for each $(v, v') \in E_s$, $v \in V_s$ and $v' \in V_s$, and for each $v \in V_s$, $f_s(v) = f(v)$ and $\nu_s(v) = \nu(v)$.

Pattern queries. A pattern query Q is a directed graph (V_Q, E_Q, f_Q, g_Q), where (1) V_Q, E_Q and f_Q are analogous to their counterparts in data graphs; and (2) for each node u in V_Q, $g_Q(u)$ is the *predicate* of u, defined as a conjunction of atomic formulas of the form $f_Q(u)$ op c, where c is a constant, and op is one of $=$, $>$, $<$, $\leqslant$ and $\geqslant$. For instance, in pattern Q_0 of Fig. 6.1, g_Q(year) = year $\geqslant$ 2011 $\wedge$ year $\leqslant$ 2013. We simply write Q as (V_Q, E_Q) or (V_Q, E_Q, f_Q).

We consider two semantics of graph pattern matching.

Subgraph queries. A match of Q in G via *subgraph isomorphism* [Ull76] is a subgraph $G'(V', E', f')$ of G that is isomorphic to Q, *i.e.,* there exists a *bijective function* h from V_Q to V' such that (a) (u, u') is in E_Q if and only if $(h(u), h(u')) \in E'$, and (b) for each $u \in V_Q$, $f_Q(u) = f'(h(u))$ and $g_Q(\nu(h(u)))$ evaluates to *true*, where $g_Q(\nu(h(u)))$ substitutes $\nu(h(u))$ for $f_Q(u)$ in $g_Q(u)$. Here $Q(G)$ is the set of all matches of Q in G.

Simulation queries. A match of Q in G via *graph simulation* [HHK95a] is a binary match relation $R \subseteq V_Q \times V$ such that (a) for each $(u, v) \in R$, $f_Q(u) = f(v)$ and $g_Q(\nu(v))$ evaluates to *true*, where $g_Q(\nu(v))$ substitutes $\nu(v)$ for $f_Q(u)$ in $g_Q(u)$; (b) for each node u in V_Q, there exists a node v in V such that (i) $(u, v) \in R$, and (ii) for any edge (u, u') in Q, there exists an edge (v, v') in G such that $(u', v') \in R$.

For any Q and G, there exists a *unique maximum* match relation R_M via graph simulation (possibly empty) [HHK95a]. Here $Q(G)$ is defined to be R_M. Simulation queries are widely used in social community analysis and social marketing [BHK+10].

Data locality. A query Q is *localized* if for any graph G that matches Q, any node u and neighbor u' of u in Q, and for any match v of u in G, there must exist a match v' of u' in G such that v' is a neighbor of v in G. Subgraph queries are localized. In contrast, simulation queries are *non-localized*.

Example 28: Consider a simulation query Q_1 and graph G_1 shown in Fig. 6.2, where G_1 matches Q_1. Then Q_1 is not localized: u_2 matches $v_2, \dots, v_{2n-2}$ and v_{2n}, but for all $k \in [2,n]$, v_{2k-2} has no neighbor in G that matches the neighbor u_3 of u_2 in Q. To decide whether u_2 matches v_2, we have to inspect all the nodes on an unbounded cycle in G_1. □

We will study bounded evaluability for subgraph queries in Sections 6.2–6.4, and then extend the results to non-localized simulation queries in Section 6.5. To formalize boundedly evaluable patterns, we first define access constraints on graphs.

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

$$S \to (l, N),$$

where $S \subseteq \Sigma$ is a (possibly empty) set of labels, l is a label in Σ, and N is a natural number.

A graph $G(V, E, f)$ *satisfies* the access constraint if

- ○ for any S-labeled set V_S of nodes in V, there exist at most N common neighbors of V_S with label l; and
- ○ there exists an *index on S for l* such that for any S-labeled set V_S in G, it finds all common neighbors of V_S labeled with l in $O(N)$-time, independent of $|G|$.

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

An access constraint is a combination of (a) *a cardinality constraint* and (b) *an index* on the labels of neighboring nodes. It tells us that for any S-node labeled set V_S, there exist a bounded number of common neighbors V_l labeled with l and moreover, V_l can be efficiently retrieved with the index.

Two special *types* of access constraints are as follows:

(1) $|S| = 0$ (*i.e.,* $\emptyset \to (l, N)$): for any G that satisfies the constraint, there exist at most N nodes in G labeled l; and

(2) $|S| = 1$ (*i.e.,* $l \to (l', N)$): for any G that satisfies the access constraint and for each node v labeled with l in G, at most N neighbors of v are labeled with l'.

Intuitively, constraints of type (1) are global cardinality constraints on all nodes labeled l, and those of type (2) state cardinality constraints on l'-neighbors of each l-labeled node.

Example 29: Constraints C1-C6 on IMDbG given in Example 27 can be expressed as access constraints φ_i (for $i \in [1,6]$):

φ_1: $(\mathsf{year}, \mathsf{award}) \to (\mathsf{movie}, 4)$; φ_4: $\emptyset \to (\mathsf{year}, 135)$;
φ_2: $\mathsf{movie} \to (\mathsf{actors/actress}, 30)$; φ_5: $\emptyset \to (\mathsf{award}, 24)$;
φ_3: $\mathsf{actor/actress} \to (\mathsf{country}, 1)$; φ_6: $\emptyset \to (\mathsf{country}, 196)$.

Here φ_2 denotes a pair $\mathsf{movie} \to (\mathsf{actors}, 30)$ and $\mathsf{movie} \to (\mathsf{actress}, 30)$ of access constraints; similarly for φ_3. Note that $\varphi_4 - \varphi_6$ are constraints of type (1); $\varphi_2 - \varphi_3$ are of type (2); and φ_1 has the general form: for any pair of year and award nodes, there are at most 4 movie nodes connected to both, *i.e.,* an award is given to at most 4 movies each year. We use $\mathcal{A}_0$ to denote the set of these access constraints. □

boundedly evaluable patterns. A pattern query Q is *boundedly evaluable under* an access schema $\mathcal{A}$ if for *all* graphs G that satisfy $\mathcal{A}$, there exists a subgraph G_Q of G such that

(a) $Q(G_Q) = Q(G)$; and
(b) G_Q can be identified in time that is determined by Q and $\mathcal{A}$ only, *not by* $|G|$.

By (b), $|G_Q|$ is also *independent of* the size $|G|$ of G. Intuitively, Q is boundedly evaluable under $\mathcal{A}$ if for all graphs G that satisfy $\mathcal{A}$, $Q(G)$ can be computed by accessing a bounded G_Q rather than the entire G, and moreover, G_Q can be efficiently accessed by using access constraints of $\mathcal{A}$.

For instance, as shown in Example 27, query Q_0 is boundedly evaluable under the access schema $\mathcal{A}_0$ of Example 29.

Discovering access constraints. From experiments with real-life data we find that many practical queries are boundedly evaluable under simple access constraints $S \to (l, N)$ when $|S|$ is at most 3. We discover access constraints as follows.
(1) Degree bounds: if each node with label l has degree at most N, then for any label l', $l \to (l', N)$ is an access constraint.
(2) Constraints of type (1): such global constraints are quite common, *e.g.,* φ_6 on IMDbG: $\emptyset \to (\mathsf{country}, 196)$.
(3) Functional dependencies (FDs): our familiar FDs $X \to A$ are access constraints of the form $X \to (A, 1)$, *e.g.,* $\mathsf{movie} \to \mathsf{year}$ is an access constraint of type (2): $\mathsf{movie} \to (\mathsf{year}, 1)$. Such constraints can be discovered by shredding a graph into relations and then using available FD discovery tools.
(4) Aggregate queries: such queries allow us to discover the semantics of the data, *e.g.,* grouping by (year, country, genre) we find $(\mathsf{year}, \mathsf{country}, \mathsf{genre}) \to (\mathsf{movie}, 1800)$, *i.e.,*

each country releases at most 1800 movies per year in each genre.

Maintaining access constraints. The indices in an access schema can be incrementally and *locally* maintained in response to changes to the data graph G. It suffices to inspect $\Delta G \cup \mathsf{Nb}_G(\Delta G)$, where ΔG is the set of nodes and edges deleted or inserted, and $\mathsf{Nb}_G(\Delta G)$ is the set of neighbors of those nodes in ΔG, regardless of how big G is.

6.2 Bounded Evaluability of Subgraph Queries

To make practical use of bounded evaluability, we first answer the following question, denoted by $\mathsf{EBnd}(\mathsf{Q}, \mathcal{A})$:

- Input: A pattern query $Q(V_Q, E_Q)$, an access schema $\mathcal{A}$.
- Question: Is Q boundedly evaluable under $\mathcal{A}$?

We start with subgraph queries. The good news is that

(a) there exists a sufficient and necessary condition, *i.e.,* a *characterization*, for deciding whether a subgraph query Q is boundedly evaluable under $\mathcal{A}$; and better still,

(b) $\mathsf{EBnd}(\mathsf{Q}, \mathcal{A})$ is decidable in low polynomial time in the size of Q and $\mathcal{A}$, independent of any data graph.

We prove these results in the rest of the section.

6.2.1 Characterizing Bounded Evaluability

The bounded evaluability of subgraph queries is characterized in terms of a notion of *coverage*, given as follows.

The *node cover* of $\mathcal{A}$ on Q, denoted by $\mathsf{VCov}(Q, \mathcal{A})$, is the set of nodes in Q computed inductively as follows:

(a) if $\emptyset \to (l, N)$ is in $\mathcal{A}$, then for each node u in Q with label l, $u \in \mathsf{VCov}(Q, \mathcal{A})$; and

(b) if $S \to (l, N)$ is in $\mathcal{A}$, then for each S-labeled set V_S in Q, if $V_S \subseteq \mathsf{VCov}(Q, \mathcal{A})$, then all common neighbors of V_S in Q that are labeled with l are also in $\mathsf{VCov}(Q, \mathcal{A})$.

Intuitively, a node u is covered by $\mathcal{A}$ if in any graph G satisfying $\mathcal{A}$, there exist a bounded number of *candidate matches* of u, and the candidates can be retrieved by using indices in $\mathcal{A}$. Obviously, (a) u is covered if its candidates are bounded by type (1) constraints. (b) If for some $\varphi = S \to (l, N)$ in $\mathcal{A}$, u is labeled with l and is a common neighbor of V_S that is covered by $\mathcal{A}$, then u is covered by $\mathcal{A}$, since its candidates are bounded (by N and the bounds on candidate matches of V_S), and can be retrieved by using the index of φ.

The *edge cover* of $\mathcal{A}$ on Q, denoted by $\mathsf{ECov}(Q,\mathcal{A})$, is the set of edges in Q defined as follows: (u_1,u_2) is in $\mathsf{ECov}(Q,\mathcal{A})$ if and only if there exist an access constraint $S \to (l,N)$ in $\mathcal{A}$ and a S-labeled set V_S in Q such that (1) u_1 (resp. u_2) is in V_S and $V_S \subseteq \mathsf{VCov}(Q,\mathcal{A})$ and (2) $f_Q(u_2) = l$ (resp. $f_Q(u_1) = l$).

Intuitively, (u_1,u_2) is in $\mathsf{ECov}(Q,\mathcal{A})$ if one of u_1 and u_2 is covered by $\mathcal{A}$ and the other has a bounded number of candidate matches by $S \to (l,N)$. Thus, we can verify their matches in a graph G by accessing a bounded number of edges.

Note that $\mathsf{VCov}(Q,\mathcal{A}) \subseteq V_Q$ and $\mathsf{ECov}(Q,\mathcal{A}) \subseteq E_Q$.

The node and edge covers characterize boundedly evaluable subgraph queries

Theorem 51: *A subgraph query Q is boundedly evaluable under an access schema $\mathcal{A}$ if and only if (iff)* $\mathsf{VCov}(Q,\mathcal{A}) = V_Q$ *and* $\mathsf{ECov}(Q,\mathcal{A}) = E_Q$. □

Proof: (1) We first show that if $\mathsf{VCov}(Q,\mathcal{A}) = V_Q$ and $\mathsf{ECov}(Q,\mathcal{A}) = E_Q$, then Q is boundedly evaluable under $\mathcal{A}$. To prove this, we first express the computation of $\mathsf{VCov}(Q,\mathcal{A})$ in chase. We then prove by induction on the length of the chasing sequence, based on the data locality of subgraph isomorphism queries.

More specifically, the computation of $\mathsf{VCov}(Q,\mathcal{A})$ for Q under $\mathcal{A}$ naturally generates a chasing sequence

$$\mathsf{VCov}_0(Q,\mathcal{A}) \overset{u_1}{\longmapsto} \ldots, \overset{u_n}{\longmapsto} \mathsf{VCov}_n(Q,\mathcal{A}),$$

such that, u_1, …, and u_n are nodes of Q; $\mathsf{VCov}_0(Q,\mathcal{A}) = \emptyset$; for each $i \in (0,n]$, $u_i \notin \mathsf{VCov}_{i-1}(Q,\mathcal{A})$, $\mathsf{VCov}_{i+1}(Q,\mathcal{A}) = \mathsf{VCov}_i(Q,\mathcal{A}) \cup \{u_i\}$, and there exists an access constraint $\varphi = S \to (l,N)$ in $\mathcal{A}$ that deduces u_i with $\mathsf{VCov}_i(Q,\mathcal{A})$ by the definition of VCov; and finally, no access constraint in $\mathcal{A}$ can be applied to $\mathsf{VCov}_n(Q,\mathcal{A})$ to deduce new node u_{n+1} anymore. Intuitively, the sequence is a procedural rephrasing of the definition of $\mathsf{VCov}(Q,\mathcal{A})$ on Q with $\mathcal{A}$. We next show that Q is boundedly evaluable under $\mathcal{A}$ if $\mathsf{VCov}_n(Q,\mathcal{A}) = V_Q$ and $\mathsf{ECov}(Q,\mathcal{A}) = E_Q$, by induction on n.

Base step. When $n = 1$, since $\mathsf{VCov}_n(Q,\mathcal{A}) = V_Q$, Q is a single node u_n. Observe that u_n can only be deduced by access constraint $\varphi_\emptyset = \emptyset \to (l,N)$, where $f_Q(u_n) = l$. Therefore, Q is boundedly evaluable via fetch with the constaint $\varphi_\emptyset$.

Induction step. Suppose when $n \leqslant k$, Q is boundedly evaluable if $\mathsf{VCov}_n(Q,\mathcal{A}) = V_Q$ and $\mathsf{ECov}(Q,\mathcal{A}) = E_Q$. Now consider a pattern graph where the chasing sequence is of length $n = k+1$. Since $\mathsf{VCov}_n(Q,\mathcal{A}) = V_Q$, Q contains $k+1$ nodes $u_1, \ldots, u_n$. Consider the subgraph Q' of Q that contains nodes $u_1, \ldots, u_{n-1}$. Let ℓ_{k+1} be the chasing sequence that deduces $\mathsf{VCov}_n(Q, \mathcal{A})$. By the definition of VCov and the chasing sequence, the prefix of ℓ_{k+1} that consists of the first k steps is also a chasing sequence of Q'. By the definition of ECov, $\mathsf{ECov}(Q', \mathcal{A}) = E_{Q'}$ if $\mathsf{ECov}(Q, \mathcal{A}) = E_Q$. By the induction hypothesis, Q' is boundedly evaluable. Thus for any data graph G, $Q'(G)$ can be computed by accessing a subgraph G_Q that is scale independent of G. By the *data locality* property of subgraph queries (see Section 6.1, for any node v in G that is a match to the unique pattern $u_{k+1} \in V_Q \setminus V_{Q'}$, there must exists another node v' in G that is a match to a pattern node u' in Q', such that u is a neighbor of u' and v is a neighbor of v'. Therefore, suppose u is connected to Q' via edges $(u'_1, u), \ldots, (u'_p, u)$ of Q, we know that there are must be matches $v'_1, \ldots, v'_p$ to $u'_1, \ldots, u'_p$, respectively, such that $v'_1, \ldots, v'_p$ can be fetched via $\mathcal{A}$ in G_Q. Since u is deduced in the final step of ℓ via an access constraint $\varphi_u = S \to (l, N)$, where $f_Q(u) = l$. Then by the definition of VCov (the semantics of single step deduction in the chasing) for each match node v to u in G, there must exist an S-labeled subset V_S of node matches to $u'_1, \ldots, u'_p$, such that v is a common neighbor of V_s. Thus v can be fetched via φ_u and V_S, *i.e.,* all nodes in the data graphs that are possible matches to Q can be fetched. By the definition of ECov, the edges between these fetched nodes can also be fetched scale independently. This means Q is boundedly evaluable under $\mathcal{A}$.

(2) We next show that if Q is boundedly evaluable under $\mathcal{A}$, then $\mathsf{VCov}(Q, \mathcal{A}) = V_Q$ and $\mathsf{ECov}(Q, \mathcal{A}) = E_Q$. We prove by induction on the number of nodes in Q.

Base step. When $|V_Q| = 1$, Q is a single node u. By the definition of VCov, there must exist an access constraint access constraint φ of form $\emptyset \to (l, N)$ in $\mathcal{A}$ with $f_Q(u) = l$. Therefore, Q can be answered by a single fetch with φ, *i.e., Q* is boundedly evaluable.

Induction step. Suppose for any subgraph query Q_k with no more than k ($k \geqslant 1$) nodes and any $\mathcal{A}$, if Q_k is boundedly evaluable under $\mathcal{A}$, then $V_k = \mathsf{VCov}(Q_k, \mathcal{A})$ and $E_k = \mathsf{ECov}(Q_k, \mathcal{A})$. We next show that for any subgraph query Q_{k+1} with $k+1$ nodes and for any access schema $\mathcal{A}$, if Q_{k+1} is boundedly evaluable under $\mathcal{A}$, then $V_{k+1} = \mathsf{VCov}(Q_{k+1}, \mathcal{A})$ and $E_{k+1} = \mathsf{ECov}(Q_{k+1}, \mathcal{A})$. Since Q_{k+1} is boundedly evaluable under $\mathcal{A}$, there must exists a subgraph Q_k of Q with k nodes such that Q_k is boundedly evaluable under $\mathcal{A}$. By the induction hypothesis, we know that $V_{Q_k} = \mathsf{VCov}(Q_k, \mathcal{A})$ and $E_{Q_k} = \mathsf{ECov}(Q_k, \mathcal{A})$. Let $\{u_{k+1}\}$ be $V_Q \setminus V_{Q_k}$ and assume *w.l.o.g.* that u_{k+1} is connected to Q_k with two edge (u_k, u_{k+1}) and (u_1, u_{k+1}). Since Q_{k+1} is boundedly evaluable, for any data graph G, all matched nodes in G to Q have to be fetched via access constraints.

By the data locality of subgraph queries, matches to u_{k+1} can only be fetched with matches to u_1 or u_k (or both). Therefore, there exist either (a) φ_1 of type (1): $\emptyset \to (l, N_1)$ with $f_Q(u_{k+1}) = l$; or (b) φ_2 of type (2): $S \to (l, N_2)$ with (i) $S = \{l'\}$, $f_Q(u_k) = l'$, and $f_Q(u_{k+1}) = l$; or (ii) $S = \{l'\}$, $f_Q(u_1) = l'$, and $f_Q(u_{k+1}) = l$; or (iii) $S = \{l', l'')$, $f_Q(u_1) = l'$, $f_Q(u_k) = l''$, and $f_Q(u_{k+1}) = l$. Otherwise the number of matches to u_{k+1} is not bounded and is not independent of $|G|$.

Consider case (a). We have $\mathsf{VCov}(Q_{k+1}, \mathcal{A}) = \mathsf{VCov}(Q_k, \mathcal{A}) \cup \{u_{k+1}\} = V_{Q_{k+1}}$. However, to *correctly* connect matches to u_{k+1} identified by index under access constraint φ_1 to matches to u_1 and u_k, there must exist access constraints that covers edge (u_1, u_{k+1}) and (u_2, u_{k+1}), to verify the correctness of the connection between the matches. That is (u_1, u_{k+1}) and (u_k, u_{k+1}) must be in $\mathsf{ECov}(Q_{k+1}, \mathcal{A})$.

Consider case (b)(i&ii). We have $\mathsf{VCov}(Q_{k+1}, \mathcal{A}) = \mathsf{VCov}(Q_k, \mathcal{A}) \cup \{u_{k+1}\} = V_{Q_{k+1}}$. For $\mathsf{ECov}(Q, \mathcal{A})$, similar to case (a), there must exist access constraints in $\mathcal{A}$ that cover edges (u_1, u_{k+1}) and (u_k, u_{k+1}) to ensure candidate matches to u_{k+1} retrieved by the index under φ_2 are correctly connected to matches retrieved for u_1 and u_k, to form a *subgraph* of G for all $G \models \mathcal{A}$.

Consider case (b)(iii). Similar to the above case, we have $\mathsf{VCov}(Q, \mathcal{A}) = V_{Q_{k+1}}$ and $\mathsf{ECov}(Q, \mathcal{A}) = E_{Q_{k+1}}$, by φ_2. □

Example 30: For query $Q_0(V_0, E_0)$ of Fig. 6.1 and access schema $\mathcal{A}_0$ of Example 29, one can verify that $\mathsf{VCov}(Q_0, \mathcal{A}_0) = V_0$ and $\mathsf{ECov}(Q_0, \mathcal{A}_0) = E_0$. From this and Theorem 51 it follows that Q_0 is boundedly evaluable under $\mathcal{A}_0$. □

6.2.2 Checking Boundedly Evaluable Subgraph Queries

Capitalizing on the characterization, we show that whether Q is boundedly evaluable under $\mathcal{A}$ can be efficiently decided.

Theorem 52: *For subgraph queries Q, $\mathsf{EBnd}(Q, \mathcal{A})$ is in*

(1) $O(|\mathcal{A}||E_Q| + ||\mathcal{A}|||V_Q|^2)$ time in general; and

(2) $O(|\mathcal{A}||E_Q| + |V_Q|^2)$ time when either

- *for each node in Q, its parents have distinct labels; or*
- *all access constraints in $\mathcal{A}$ are of type (1) or (2).*

□

Here $|\mathcal{A}|$ denotes the total length of constraints in $\mathcal{A}$, $||\mathcal{A}||$ is the number of constraints in $\mathcal{A}$, and a node u' is a *parent* of u in Q if there exists an edge from u' to u in Q.

```
Algorithm EBChk
Input: A subgraph query $Q$ and an access schema $\mathcal{A}$.
Output: "yes" if $Q$ is boundedly evaluable and "no" otherwise.
1.  for each $S \rightarrow (l,N)$ in $\mathcal{A}$ ($S \neq \emptyset$) do
2.      find all $\bar{V}_S^u \mapsto (u,N)$ in $Q$ and add them to $\Gamma$; /*$f(u) = l$*/
3.  $\mathcal{B} := \{v \in V_Q \mid \emptyset \rightarrow (f_Q(v),N)$ is in $\mathcal{A}\}$;
4.  $\mathcal{C} := \mathcal{B}$; /*Initialize $\mathsf{VCov}(Q,\mathcal{A})$*/
5.  $\mathsf{InitAuxi}(L,ct)$; /*Initialize auxiliary structures*/
6.  while $\mathcal{B}$ is not empty do
7.      $v = \mathcal{B}.\mathsf{pop}()$;
8.      for each $\phi$ in $L[v]$ do
9.          $\mathsf{Update}\ (ct[\phi])$; /*Update counter $ct[\phi]$*/
10.         if $ct[\phi] = 0$ and $u \notin \mathcal{C}$ do /*suppose $\phi$: $\bar{V}_S^u \mapsto (u,N)$*/
11.             $\mathcal{B} := \mathcal{B} \cup \{u\}$; $\mathcal{C} := \mathcal{C} \cup \{u\}$;
12. if $V_Q \subseteq \mathcal{C}$ and all edges in $Q$ are in $\mathsf{ECov}(Q,\mathcal{A})$ then
13.     return "yes";
14. return "no";
```

Figure 6.3: Algorithm EBChk

Algorithm. We prove Theorem 52 by providing a checking algorithm. The algorithm is denoted by EBChk and shown in Fig. 6.3. Given a subgraph query $Q(V_Q,E_Q)$ and an access schema $\mathcal{A}$, it checks whether (a) $V_Q \subseteq \mathsf{VCov}(Q,\mathcal{A})$ and (b) $E_Q \subseteq \mathsf{ECov}(Q,\mathcal{A})$; it returns "yes" if so, by Theorem 51.

To check these conditions, we actualize $\mathcal{A}$ on Q: for each $S \rightarrow (l,N)$ in $\mathcal{A}$ ($S \neq \emptyset$), and each node u in Q with $f_Q(u) = l$, the *actualized constraint* is $\bar{V}_S^u \mapsto (u,N)$, where $\bar{V}_S^u$ is the maximum set of neighbors of u in Q such that (a) there exists a S-labeled set $V_S \subseteq \bar{V}_S^u$ and (b) for each u' in $\bar{V}_S^u$, $f_Q(u') \in S$.

Actualized constraints help us deduce $\mathsf{VCov}(Q,\mathcal{A})$: a node u of Q is in $\mathsf{VCov}(Q,\mathcal{A})$ if and only if either

- there exists $\emptyset \rightarrow (l,N)$ in $\mathcal{A}$ and $f_Q(u) = l$; or
- $\bar{V}_S^u \mapsto (u,N)$ and there exists a S-labeled set of Q that is a subset of $\bar{V}_S^u \cap \mathsf{VCov}(Q,\mathcal{A})$.

When $\mathsf{VCov}(Q,\mathcal{A})$ is in place, we can easily check whether $E_Q \subseteq \mathsf{ECov}(Q,\mathcal{A})$

by definition and using the actualized constraints, without explicitly computing $\mathsf{ECov}(Q, \mathcal{A})$.

We next present the details of algorithm EBChk.

Auxiliary structures. EBChk uses three auxiliary structures.

(a) It maintains a set $\mathcal{B}$ of nodes in Q that are in $\mathsf{VCov}(Q, \mathcal{A})$ but it remains to be checked whether other nodes can be deduced from them. Initially, $\mathcal{B}$ includes nodes whose labels are covered by type (1) constraints in $\mathcal{A}$ (line 3). EBChk uses $\mathcal{B}$ to control the **while** loop (lines 5-10): it terminates when $\mathcal{B} = \emptyset$, *i.e.,* all candidates for $\mathsf{VCov}(Q, \mathcal{A})$ are found.

(b) For each node v, EBChk uses an inverted index $L[v]$ to store all actualized constraints $\bar{V}_S^u \mapsto (u, N)$ such that $v \in \bar{V}_S^u$. That is, $L[v]$ indexes those constraints in Γ that can be used on v.

(c) For each actualized constraint $\phi = \bar{V}_S^u \mapsto (u, N)$, EBChk maintains a set $ct[\phi]$ to keep track of those labels of S that are *not covered* by nodes in $\bar{V}_S^u \cap \mathsf{VCov}(Q, \mathcal{A})$ yet. Initially, $ct[\phi] = S$. When $ct[\phi]$ is empty, EBChk concludes that there is a S-labeled subset of $\bar{V}_S^u$ covered by $\mathsf{VCov}(Q, \mathcal{A})$, and thus deduces that u should also be in $\mathsf{VCov}(Q, \mathcal{A})$ (line 10).

Using these, EBChk works in the following two steps.

(1) Computing Γ. It finds all actualized constraints of $\mathcal{A}$ on Q and puts them in Γ (lines 1-2). This can be done by scanning all nodes of Q and their neighbors for each access constraint in $\mathcal{A}$. Observe that there are at most $||\mathcal{A}|||V_Q|$ actualized constraints in Γ, *i.e.,* Γ is bounded by $O(||\mathcal{A}|||E_Q|)$.

(2) Computing $\mathsf{VCov}(Q, \mathcal{A})$, stored in a variable $\mathcal{C}$. After initializing auxiliary structures as described above via procedure InitAuxi (omitted; lines 3-5), EBChk processes nodes in $\mathcal{B}$ one by one (lines 6-11). For each $u \in \mathcal{B}$ and each actualized constraint $\phi = \bar{V}_S^v \mapsto (v, N)$ in $L[u]$, it updates the set $ct[\phi]$ by removing label $f_Q(u)$ by procedure Update (omitted; line 9). When $ct[\phi] = \emptyset$, *i.e.,* there exists a S-labeled subset in $\bar{V}_S^v$ that is covered by $\mathcal{C}$, EBChk adds u to $\mathcal{C}$ and $\mathcal{B}$ (lines 10-11). When $\mathcal{B}$ is empty, *i.e.,* all nodes have been inspected, EBChk checks whether $V_Q \subseteq \mathsf{VCov}(Q, \mathcal{A})$ and whether all edges are covered by $\mathsf{ECov}(Q, \mathcal{A})$. It returns “yes” if so (lines 12-13).

Example 31: Given subgraph query Q_0 of Fig. 6.1 and access schema $\mathcal{A}_0$ of Example 29, EBChk first computes the set Γ of actualized constraints: $\phi_1 = (u_1, u_2) \mapsto (u_3, 4)$, $\phi_2 = u_3 \mapsto (u_4/u_5, 30)$, and $\phi_3 = u_4/u_5 \mapsto (u_6, 1)$. It then sets both $\mathcal{B}$ and $\mathcal{C}$ to be $\{u_1, u_2, u_6\}$, and initializes $ct[\phi_1], \ldots, ct[\phi_3]$ and lists $L[u_1], \ldots, L[u_6]$ accordingly. EBChk then pops u_1 and u_2 off from $\mathcal{B}$ and finds that u_3 can be deduced. Thus it adds u_3 to $\mathcal{B}$ and $\mathcal{C}$. It then pops u_3 off from $\mathcal{B}$, processes u_4 and u_5, and confirms that u_4 and u_5 should be included in $\mathcal{C}$. At this point, it finds that $\mathcal{C}$ contains all the nodes in Q and moreover, each edge in Q is also covered by at least one access constraint in $\mathcal{A}_0$. Thus it returns "yes". □

Correctness & Complexity. The correctness of EBChk follows from Theorem 51 and the properties of actualized constraints stated above. We next analyze the time complexity of EBChk.

(1) General case. Observe the following. (a) Computing Γ is in $O(|\mathcal{A}||E_Q|)$ time, since for each φ in $\mathcal{A}$, we can find all actualized constraints of φ in $O(\Sigma_{v \in V_Q} \mathsf{deg}(v)|\varphi|) = O(|\varphi||E_Q|)$ time, where $\mathsf{deg}(v)$ is the number of neighbors of v. (b) Computing $\mathsf{VCov}(Q, \mathcal{A})$ takes $O(||\mathcal{A}|||V_Q|^2)$ time. For each φ in $\mathcal{A}$, the sets $ct(\phi)$ for all corresponding actualized constraints ϕ in Γ are updated in time $O(\Sigma_{v \in V_Q}(\mathsf{deg}(v)^2)) = O(|V_Q|^2)$. As each ϕ in Γ is processed once, the total time is bounded by $O(||\mathcal{A}|||V_Q|^2)$. (c) The checking of lines 12-13 takes $O(|\mathcal{A}||E_Q| + |V_Q|^2)$ time. Thus, EBChk takes $O(|\mathcal{A}||E_Q| + ||\mathcal{A}|||V_Q|^2 + |V_Q|^2) = O(|\mathcal{A}||E_Q| + ||\mathcal{A}|||V_Q|^2)$ time.

(2) Special cases. We next show that EBChk can be optimized to $O(|\mathcal{A}||E_Q| + |V_Q|^2)$ time for each of the two special cases given in Theorem 52. The idea is to use a counter $n[\phi]$ instead of $ct[\phi]$ in EBChk such that $n[\phi]$ always equals $|ct[\phi]|$. This does not hurt the correctness since in the special cases, each time when we update $ct[\phi]$, we remove a distinct label. With this new auxiliary structure, step (b) in the analysis above is in $O(||\mathcal{A}|||E_Q|)$ time in total since the counters are updated $O(||\mathcal{A}||(\Sigma_{v \in V_Q} \mathsf{deg}(v))) = O(||\mathcal{A}|||E_Q|)$ times in total, and each updates takes $O(1)$ time: it just decreases $n[\phi]$ by 1. The correctness for this optimization is that each time we update $ct[\phi]$, we removes a distinct label. And because of this, EBChk can use the counter $n[\phi]$ instead of the set $ct[\phi]$.

6.3 Generating Bounded Query Plans

After a subgraph query $Q(V_Q, E_Q)$ is found boundedly evaluable under an access schema $\mathcal{A}$, we need to generate a "good" query plan for Q that, given any (big) graph G, computes $Q(G)$ by fetching a small G_Q such that $Q(G) = Q(G_Q)$ and $|G_Q|$ is determined by Q and $\mathcal{A}$, independent of $|G|$.

The main results of this section are as follows:

- a notion of worst-case optimality for query plans; and

- an algorithm to generate worst-case-optimal query plans in $O(|V_Q||E_Q||\mathcal{A}|)$ time.

Below we first formalize query plans and define the worst-case optimality. We then present the algorithm.

Query plans. A *query plan* $\mathcal{P}$ for Q under $\mathcal{A}$ is a sequence of *node fetching* operations of the form $\mathsf{fetch}(u, V_S, \varphi, g_Q(u))$, where u is a l-labeled node in Q, V_S denotes a S-labeled set of Q, φ is a constraint $\varphi = S \to (l, N)$ in $\mathcal{A}$, and $g_Q(u)$ is the predicate of u (refer to Section 6.1 for the definition).

On a graph G, the operation is to retrieve a set $\mathsf{cmat}(u)$ of *candidate matches* for u from G: given V_S that was retrieved from G earlier, it fetches common neighbors of V_S from G that (i) are labeled with l and (ii) satisfy the predicate $g_Q(u)$ of u. These nodes are fetched by using the index of φ and are stored in $\mathsf{cmat}(u)$. In particular, when $S = \emptyset$, the operation fetches all l-labeled nodes in G as $\mathsf{cmat}(u)$ for u.

The operations $\mathsf{fetch}_1\mathsf{fetch}_2\cdots\mathsf{fetch}_n$ in $\mathcal{P}$ are executed one by one, in this order. There may be multiple operations for the same node u in Q, each fetching a set V_i^u of candidates for u from G. We will ensure that for fetch_i and fetch_j for u, V_j^u has less nodes than V_i^u if $i < j$, and we say that fetch_j *reduces* $\mathsf{cmat}(u)$ fetched by fetch_i. We denote V_k^u by V_u, where fetch_k is the last operation for u in $\mathcal{P}$, *i.e.,* it fetches the smallest $\mathsf{cmat}(u)$ for u.

Building G_Q. Intuitively, $\mathcal{P}$ tells us what nodes to retrieve from G. From the data fetched by $\mathcal{P}$, a subgraph $G_Q(V_{\mathcal{P}}, E_{\mathcal{P}})$ is built and used to compute $Q(G)$. More specifically, (a) $V_{\mathcal{P}} = \bigcup_{u \in Q} V_u$, *i.e.,* it contains maximally reduced $\mathsf{cmat}(u)$ for each node u in Q; and (b) $E_{\mathcal{P}}$ consists of the following: for each node pairs (v, v') in $V_u \times V_{u'}$, if (u, u') is an edge in Q, we check whether (v, v') is an edge in G and if so, include it in $E_{\mathcal{P}}$. This is done by accessing a bounded amount of data: we first find $\varphi_{u'} = S \to (f_Q(u'), N)$ in $\mathcal{A}$ and a S-labeled set V_s such that $v \in V_S$; we then fetch common neighbors of V_S by using the index of $\varphi_{u'}$ and check whether v' is one of them. As Q is boundedly evaluable under $\mathcal{A}$ (*i.e.,* $\mathsf{ECov}(Q, A) = E_Q$), if (v, v') is an edge in G then such $\varphi_{u'}$ and V_S exist.

Bounded plans. We say that a query plan $\mathcal{P}$ for Q under $\mathcal{A}$ is *boundedly evaluable* if for all $G \models \mathcal{A}$, it builds a subgraph G_Q of G such that (a) $Q(G_Q) = Q(G)$ and (b) the time for fetching data from G by *all operations* in $\mathcal{P}$ depends on $\mathcal{A}$ and Q only. That is, $\mathcal{P}$ fetches a bounded amount of data from G and builds G_Q from it. By (b), $|G_Q|$ is *independent of* $|G|$.

Optimality. We naturally want an *optimal* plan $\mathcal{P}$ that finds us a minimum G_Q, *i.e.,* for each graph $G \models \mathcal{A}$, G_Q identified by $\mathcal{P}$ has the smallest size among all subgraphs identified by any boundedly evaluable query plans. Unfortunately, the result below shows that this is impossible.

Theorem 53: *There exists no query plan that is both boundedly evaluable and optimal for all graphs $G \models \mathcal{A}$.* $\square$

Proof: Note that the query plan generated by algorithm QPlan is already instance-$\langle Q, \mathcal{A}\rangle$ optimal, which shows algorithm QPlan is an instance optimal algorithm in terms of worst-case $|G_Q|$. We show there exist no better algorithms: there exists no instance-$\langle Q, \mathcal{A}, G\rangle$ optimal algorithm.

The result naturally follows from the fact the G is not part of the input when generating query plans $\mathcal{P}$: the input consists of Q and $\mathcal{A}$ only and thus, each pair of Q and $\mathcal{A}$ uniquely determine a query plan $\mathcal{P}$ which extracts G_Q from all $G \models \mathcal{A}$. Thus there exists no instance-$\langle Q, \mathcal{A}, G\rangle$ optimal effectviely bounded query plans. Indeed, one can verify the following: there exists a pair of Q and $\mathcal{A}$, such that one can construct two data graphs G and G' that both satisfy $\mathcal{A}$ and moreover, the query plan determined by Q and $\mathcal{A}$, denoted by $\mathcal{P}_{Q,\mathcal{A}}$, cannot find optimal G_Q from both G and G': if $\mathcal{P}_{Q,\mathcal{A}}(G)$ is the minimum G_Q for Q on G such that $Q(G) = Q(G')$, then there exists $G'_Q \subseteq G$ such that $|G'_Q| \leqslant |\mathcal{P}_{Q,\mathcal{G}}(G')|$ and moreover, $Q(G'_Q) = Q(G)$.

An example Q and $\mathcal{A}$ is as follows. Define $Q(V_Q,E_Q)$: V_Q = $\{u_1,\ u_2,\ u_3\}$, E_Q = $\{(u_1,u_3),\ (u_2,u_3)\}$. $f(u_1) = A$, $f_{u_2} = B$, $f_{u_3} = C$. Let $\mathcal{A}$ consists of 4 access constraints: $\varphi_1 = \emptyset \to (A, 10)$, $\varphi_2 = \emptyset \to (B, 10)$, $\varphi_3 = A \to (C, 10)$, and $\varphi_4 = B \to (C, 10)$. There are only two *different* boundedly evaluable query plans: $\mathcal{P}_1 = s_1s_2s_3s_4$, where s_1 = $\mathsf{fetch}(u_1, \emptyset, \varphi_1, \mathit{true})$, s_2 = $\mathsf{fetch}(u_2, \emptyset, \varphi_2, \mathit{true})$, s_3 = $\mathsf{fetch}(u_3, \{u_1\}, \varphi_3, \mathit{true})$. s_4 = $\mathsf{fetch}(u_3, \{u_2\}, \varphi_4, \mathit{true})$. $\mathcal{P}_2 = s_1s_2s_4s_3$. Now consider G_1 and G_2 that both satisfy $\mathcal{A}$ as follows. G_1 is similar to Q except that there are 10 C-labeled nodes all connected to B and only one of them is connected to A. G_2 instead has 10 C-labeled nodes all connected to A but 1 of them connected to B. It is easy to see that $\mathcal{P}_1$ get an minimum G_Q in G_1 but not in G_2 and vice-versa for $\mathcal{P}_2$. This shows no algorithm is instance-$\langle Q, \mathcal{A}, G\rangle$ optimal, *i.e.,* no boundedly evaluable query plan that is optimal for each Q and $\mathcal{A}$ on all $G \models \mathcal{A}$. $\square$

This motivates us to introduce worst-case optimality. An boundedly evaluable query plan $\mathcal{P}$ for Q under $\mathcal{A}$ is *worst-case optimal* if for any other boundedly evaluable query plan $\mathcal{P}'$ for Q under $\mathcal{A}$, $\mathsf{max}_{G\models\mathcal{A}}|G_Q| \leqslant \mathsf{max}_{G\models\mathcal{A}}|G'_Q|$, where G_Q and G'_Q are subgraphs identified by $\mathcal{P}$ and $\mathcal{P}'$, respectively.

That is, given any Q and $\mathcal{A}$, for *all* $G \models \mathcal{A}$, the *largest* subgraph G_Q identified by $\mathcal{P}$

```
Algorithm QPlan
Input: An boundedly evaluable subgraph query $Q$, access schema $\mathcal{A}$.
Output: A worst-case optimal and boundedly evaluable query plan $\mathcal{P}$.
1.  Build actualized graph $Q_\Gamma(V_\Gamma, E_\Gamma)$ from $Q$ and $\Gamma$;
2.  for each $u$ in $V_\Gamma$ do
3.      $\mathsf{size}[u] := +\infty$; $\mathsf{sn}[u] := \mathit{false}$;
4.      if there exists $\varphi = \emptyset \to (l, N)$ in $\mathcal{A}$ with $f_Q(u) = l$ do
5.          append $\mathsf{fetch}(u, \mathit{nil}, \varphi, g_Q(u))$ to $\mathcal{P}$;
6.          $\mathsf{sn}[u] := \mathit{true}$; $\mathsf{size}[u] := N$;
7.  while there exists $u$ in $V_\Gamma$ such that $\mathsf{check}(u) = \mathit{true}$ do
8.      $(V_u, \varphi_u, \mathsf{size}[u], \mathsf{sn}[u]) := \mathsf{ocheck}(u)$;
9.      append $\mathsf{fetch}(u, V_u, \varphi_u, g_Q(u))$ to $\mathcal{P}$;
10. return $\mathcal{P}$;
```

Figure 6.4: Algorithm QPlan

is no larger than the worst-case subgraphs identified by any other boundedly evaluable query plans.

Worst-case optimal query plans are within reach in practice.

Theorem 54: *There exists an algorithm that, given any boundedly evaluable subgraph query Q under an access schema $\mathcal{A}$, finds a query plan that is both boundedly evaluable and worst-case optimal for Q under $\mathcal{A}$, in $O(|V_Q||E_Q||\mathcal{A}|)$ time.* □

Algorithm. We prove Theorem 54 by giving such an algorithm, denoted by QPlan and shown in Fig. 6.4. The algorithm inspects each node u of Q, finds an access constraint φ in $\mathcal{A}$ such that its index can help us retrieve candidates $\mathsf{cmat}(u)$ for u from an input graph G, generates a fetching operation accordingly, and stores it in a list $\mathcal{P}$. It then iteratively reduces $\mathsf{cmat}(u)$ for each u in Q to optimize $\mathcal{P}$, until $\mathcal{P}$ cannot be further improved.

The algorithm uses the following structures.

(1) An *actualized graph* $Q_\Gamma(V_\Gamma, E_\Gamma)$, which is a directed graph constructed from Q and the set Γ of all actualized constraints of $\mathcal{A}$ on Q (see Section 6.2). Here (a) $V_\Gamma = V_Q$; and (b) for any two nodes u_1 and u_2 in V_Γ, (u_1, u_2) is in E_Γ iff there exists a constraint $\bar{V}_S \mapsto (u_2, N)$ in Γ such that u_1 is in $\bar{V}_S$. Intuitively, Q_Γ represents deduction relations for nodes in V_Q, and guides us to extract candidate matches for Q.

(2) For each node u in Q, a counter $\mathsf{size}[u]$ to store the cardinality of $\mathsf{cmat}(u)$, and a Boolean flag $\mathsf{sn}[u]$ to indicate whether the fetching operations in current $\mathcal{P}$ can find $\mathsf{cmat}(u)$.

With these structures, algorithm QPlan works as follows. It first builds actualized graph Q_Γ (line 1), and initializes $\mathsf{size}[u] = +\infty$ and $\mathsf{sn}[u] = \mathit{false}$ for all the nodes u in Q_Γ (lines 2-3). It then finds nodes u_0 for which $\mathsf{cmat}(u)$ can be retrieved by using the index specified in some type (1) constraints $\emptyset \to (l,N)$ in $\mathcal{A}$ (lines 4-6). For each u_0, QPlan adds a fetching operation to $\mathcal{P}$ and sets $\mathsf{sn}[u_0] = \mathit{true}$ and $\mathsf{size}[u_0] = N$.

After the initialization, QPlan recursively processes nodes u of Q to retrieve or reduce their $\mathsf{cmat}(u)$ (lines 7-9), starting from those nodes u_0 identified in line 4. It picks the next node u by a function check (omitted). Here $\mathsf{check}(u)$ does the following: it (i) finds the set V_u^p of parents of u in Q_Γ such that $\mathsf{sn}[v] = \mathit{true}$ for all $v \in V_u^p$, (ii) selects a subset V_u of V_u^p such that V_u forms a S-labeled set for some constraint $\varphi_u = S \to (f_Q(u),N)$ in $\mathcal{A}$, and moreover, $N * \Pi_{v\in V_u}\mathsf{size}[v]$ is minimum among all such S-labeled sets of u; and (iii) returns true if $N * \Pi_{v\in V_u}\mathsf{size}[v] < \mathsf{size}[u]$. If $\mathsf{check}(u) = \mathit{true}$, QPlan sets $\mathsf{size}[u] = N * \Pi_{v\in V_u}\mathsf{size}[v]$ and $\mathsf{sn}(u) = \mathit{true}$ by function ocheck (omitted), and adds a fetching operation to $\mathcal{P}$ for u using φ_u and V_u. It proceeds until for no u in Q, $\mathsf{check}(u) = \mathit{true}$ (line 7). At this point, QPlan returns $\mathcal{P}$ (line 10).

Example 32: Given query Q_0 of Fig. 6.1 and access schema $\mathcal{A}_0$ of Example 29, QPlan finds $\mathcal{P}$ as follows. Using the actualized constraints Γ of $\mathcal{A}_0$ on Q_0 (see Example 31), it first builds Q_Γ, which is the same as Q_0 except the directions of the edges (u_3,u_1) and (u_3,u_2) are reversed. Using type (1) constraints in $\mathcal{A}_0$, QPlan adds $\mathsf{fetch}_1(u_1,\mathit{nil},\varphi_5,\mathit{true})$, $\mathsf{fetch}_2(u_2,\mathit{nil},\varphi_4,\mathsf{year} \geqslant 2011 \wedge \mathsf{year} \leqslant 2013)$ and $\mathsf{fetch}_3(u_6,\mathit{nil},\varphi_6,\mathit{true})$ to $\mathcal{P}$. In the **while** loop, it finds $\mathsf{check}(u_3) = \mathit{true}$ and adds $\mathsf{fetch}_4(u_3,\{u_1,u_2\}, \varphi_1,\mathit{true})$ to $\mathcal{P}$. As a consequence of fetch_4, it finds that $\mathsf{check}(u_4)$ and $\mathsf{check}(u_5)$ become true and thus adds $\mathsf{fetch}_5(u_4,\{u_3\},\varphi_2, \mathit{true})$ and $\mathsf{fetch}_6(u_5,\{u_4\},\varphi_2,\mathit{true})$ to $\mathcal{P}$. Now $\mathcal{P}$ cannot be further improved, and it returns $\mathcal{P}$ with 6 fetching operations,

We next show how this $\mathcal{P}$ identifies G_Q from the IMDbG graph G_0 of Example 27 for Q_0. (a) It executes its fetching operations one by one, and retrieves $\mathsf{cmat}(u)$ from G_0 for u ranging over u_1–u_6, with at most 24, 3, 288, 8640, 8640 and 196 nodes, respectively. These are treated as the nodes of G_Q, no more than 17791 in total. (b) It then adds edges to G_Q. For each $(v_3,v_1) \in \mathsf{cmat}(u_3) \times \mathsf{cmat}(u_1)$, it checks whether (v_3,v_1) is an edge in G_0 by using $\mathsf{cmat}(u_1)$, $\mathsf{cmat}(u_2)$ and $\mathsf{cmat}(u_3)$, and the index of φ_1 of $\mathcal{A}_0$, as suggested by fetching operation fetch_4 for u_3 given above. If so, (v_3,v_1) is included in G_Q. This checks $24 \times 3 \times 4$ neighbors of $\mathsf{cmat}(u_3)$ in the worst case. Similarly, it examines at most 288, 8640, 8640, 8640 and 8640 candidates matches in G_0 for edges (u_3, u_2), (u_3, u_4), (u_3, u_5), (u_4, u_6) and (u_4, u_6) in Q_0, respectively. This yields at most 34848 edges in G_Q in total. Note that query plan $\mathcal{P}$ is exactly the one described in Example 27, and accesses at most 17923 nodes and 35136 edges in total. Only *part of the data* accessed by $\mathcal{P}$ is included in G_Q for answering Q. □

Correctness & Complexity. For the correctness of QPlan, observe the following about the query plan $\mathcal{P}$ generated for Q and $\mathcal{A}$. (1) *$\mathcal{P}$ is boundedly evaluable*: indeed, (a) the total amount of data fetched by $\mathcal{P}$ is decided by $\mathcal{A}$ and Q since $\mathcal{P}$ only uses indices in $\mathcal{A}$ to retrieve data; and (b) $Q(G_Q) = Q(G)$ since G_Q includes all candidate matches from G for nodes and edges in Q. By the data locality of subgraph queries, if a node v in G matches a node u in Q, then for any neighbor u' of u in Q, matches of u' must be neighbors of v in G. That is why $\mathsf{cmat}(u)$ collects candidate node matches from neighbors; similarly for edges. (2) *$\mathcal{P}$ is worst-case optimal*: since the **while** loop reduces $|\mathsf{cmat}(u)|$ to be the minimum.

To see that QPlan is in $O(|V_Q||E_Q||\mathcal{A}|)$ time, observe the following. (1) Line 1 is in $O(|\mathcal{A}||E_Q|)$ time. (2) The **for** loop (lines 2-6) is in $O(|V_Q|)$ time by using the inverted indices. (3) The **while** loop (lines 7-9) iterates $|V_Q|^2$ times, since for each node u in Q, (a) $\mathsf{cmat}(u)$ is reduced only if $\mathsf{cmat}(u')$ is reduced for its "ancestors" u' in Q_Γ, $|V_Q| - 1$ times at most, by the definition of $\mathsf{size}[u]$ and check (*i.e.,* $\mathsf{size}[u]$ remains larger than $\mathsf{size}[u']$), and (b) each reduction to $\mathsf{cmat}(u')$ requires us to check once whether $\mathsf{cmat}(u)$ is also reduced as a consequence. In each iteration, $\mathsf{check}(u)$ and $\mathsf{ocheck}(u)$ take $O(\mathsf{deg}(u)|\mathcal{A}|)$ time. As $O(|V_Q| * \Sigma_{u \in V_Q} \mathsf{deg}(u)|\mathcal{A}|) = O(|V_Q||E_Q||\mathcal{A}|)$, the **while** loop takes $O(|V_Q||E_Q||\mathcal{A}|)$ time in total.

6.4 Making Queries Instance Bounded

Consider a frequent query load $\mathcal{Q}$, such as a *finite* set of parameterized queries as found in recommendation systems. If some queries Q in $\mathcal{Q}$ are not boundedly evaluable under an access schema $\mathcal{A}$, can we still compute $Q(G)$ in a big graph G? The main conclusion of this section is positive: one can often make all queries in $\mathcal{Q}$ instance-bounded in G and answer them in G by accessing a bounded amount of data.

Extending access schemas. The idea is to extend $\mathcal{A}$ such that its indices suffice to help us fetch bounded subgraphs of G for answering $\mathcal{Q}$. Consider a *constant* M. An

M-bounded extension $\mathcal{A}_M$ of $\mathcal{A}$ includes all access constraints in $\mathcal{A}$ and additional access constraints of types (1) and (2) (see Section 6.1):

$$\text{Type (1): } \emptyset \rightarrow (l', N) \qquad\qquad \text{Type (2): } l \rightarrow (l', N)$$

such that $N \leqslant M$. Note that $\mathcal{A}_M$ is also an access schema.

Instance-bounded patterns. Consider $G \models A_M$. A set $\mathcal{Q}$ of pattern queries is *instance-bounded in G* under $\mathcal{A}_M$ if for all $Q \in \mathcal{Q}$, there exists a subgraph G_Q of G such that

(a) $Q(G_Q) = Q(G)$; and

(b) G_Q can be found in time determined by $\mathcal{A}_M$ and Q *only*.

As a result of (b) and the use of constant M, $|G_Q|$ is a function of $\mathcal{A}$, Q and M. As opposed to bounded evaluability, instance boundedness aims to process *a finite set $\mathcal{Q}$* of queries on *a particular instance G* by accessing a bounded amount of data.

Given these, we answer $\mathcal{Q}$ in a big G as follows. If some queries in $\mathcal{Q}$ are not boundedly evaluable under $\mathcal{A}$, we extend $\mathcal{A}$ to $\mathcal{A}_M$ by adding simplest access constraints such that all queries in $\mathcal{Q}$ are instance-bounded in G under $\mathcal{A}_M$.

Proposition 55: *For any finite set $\mathcal{Q}$ of subgraph queries, access schema $\mathcal{A}$ and graph $G \models \mathcal{A}$, there exist M and an M-bounded extension $\mathcal{A}_M$ under which $\mathcal{Q}$ is instance-bounded in G.* □

That is, additional access constraints of types (1) and (2) suffice to make $\mathcal{Q}$ instance-bounded in G.

Proof: Given any set $\mathcal{Q}$ of subgraph queries, we can make $\mathcal{Q}$ instance-bounded in a data graph G by using $L_{\mathcal{Q}}$ type (1) access constraints to indexing all labeled nodes, and using $C^2_{L_{\mathcal{Q}}}$ type (2) access constraints to indexing all possible edges with those labels. Thus the total number of access constraints is $\frac{L_{\mathcal{Q}}(L_{\mathcal{Q}}-1}{2} + L_{\mathcal{Q}} = \frac{L_{\mathcal{Q}}(L_{\mathcal{Q}}+1)}{2}$. □

Resource-bounded extensions. Proposition 55 always holds when M is sufficiently large. When M is a small predefined bound indicating our constrained resources, we have to answer the following question, denoted by $\mathsf{EEP}(\mathcal{Q}, \mathcal{A}, M, G)$:

- Input: A finite set $\mathcal{Q}$ of subgraph queries, an access schema $\mathcal{A}$, a natural number M, and a graph $G \models \mathcal{A}$.
- Question: Does there exist a M-bounded extension $\mathcal{A}_M$ of $\mathcal{A}$ such that $\mathcal{Q}$ is instance-bounded in G under $\mathcal{A}_M$?

This problem is decidable in PTIME.

Theorem 56: $\mathsf{EEP}(\mathcal{Q},\mathcal{A},M,G)$ *is in* $O(|G| + (|\mathcal{A}| + |\mathcal{Q}|)|E_{\mathcal{Q}}| + (||\mathcal{A}|| + |\mathcal{Q}|)|V_{\mathcal{Q}}|^2)$ *time, where* $|G| = |V| + |E|$, $|E_{\mathcal{Q}}| = \sum_{Q\in\mathcal{Q}}|E_Q|$, $|V_{\mathcal{Q}}| = \sum_{Q\in\mathcal{Q}}|V_Q|$ *and* $|\mathcal{Q}| = |E_{\mathcal{Q}}| + |V_{\mathcal{Q}}|$. □

For a frequent query load $\mathcal{Q}$, we identify A_M; if $\mathcal{A}_M$ exists, we build additional indices on G and make $G \models \mathcal{A}_M$, as *preprocessing* offline. We can then repeatedly instantiate and process query templates of $\mathcal{Q}$ by accessing a bounded amount of data in G, and *incrementally maintain* indices in response to changes to G. Note that real-life queries are typically small.

We prove Theorem 56 by giving a checking algorithm. The algorithm, denoted by EEChk, consists of two steps.

Step (1) (Maximum M-bounded extension): Find all types (1) and (2) constraints $\emptyset \to (l',N)$ and $l \to (l',N)$ on G for all labels l and (l,l') that are in *both* Q and G, such that $N \leqslant M$ and G satisfies their corresponding cardinality constraints. Let $\mathcal{A}_M$ include all these constraints and all those in $\mathcal{A}$.

Step (2) (Checking): Check whether $\mathcal{Q}$ is instance-bounded in G under $\mathcal{A}_M$ by using a mild revision of $\mathsf{EBChk}(Q,\mathcal{A}_M)$ (see Section 6.2) for each $Q \in \mathcal{Q}$; return "yes" if $\mathsf{EBChk}(Q,\mathcal{A}_M)$ returns "yes" for all Q in $\mathcal{Q}$, and "no" otherwise.

Example 33: Consider a given bound $M = 150$, the IMDbG graph G_0 of Example 27, a set $\mathcal{Q}$ with only Q_0 of Fig. 6.1, and an access schema $\mathcal{A}$ consisting of all constraints in $\mathcal{A}_0$ of Example 29 except φ_4 and φ_5. Given these, EEChk finds a M-bounded extension $\mathcal{A}_M$ of $\mathcal{A}$. (1) It finds, among others, that G satisfies the cardinality constraints of two type 1 access constraints $\varphi_4 = \emptyset \to (\mathsf{year}, 135)$ and $\varphi_5 = \emptyset \to (\mathsf{award}, 24)$, and $135 < M$ and $24 < M$. It extends $\mathcal{A}$ by including φ_4 and φ_5, yielding $\mathcal{A}_M$. (2) It then invokes $\mathsf{EBChk}(Q,\mathcal{A}_M)$ and confirms that $\mathcal{Q}$ is instance-bounded in G under $\mathcal{A}_M$. □

Correctness & Complexity. The correctness of EEChk is ensured by the following. (1) If there exists $\mathcal{A}'_M$ such that $\mathcal{Q}$ is instance-bounded in G under $\mathcal{A}'_M$, then $\mathcal{Q}$ is instance-bounded in G under $\mathcal{A}_M$ for $\mathcal{A}'_M \subseteq \mathcal{A}_M$; hence it suffices to consider the maximum M-bounded extension $\mathcal{A}_M$ of $\mathcal{A}$. (2) Checking instance boundedness is a mild revision of $\mathsf{EBChk}(Q,\mathcal{A}_M)$, with the same complexity stated in Theorem 52.

For the complexity, observe that Step (1) of EEChk is in $O(|G|)$ time, $|\mathcal{A}_M|$ and $||\mathcal{A}_M||$ are bounded by $|\mathcal{A}| + |\mathcal{Q}|$ and $||\mathcal{A}|| + |\mathcal{Q}|$, respectively. Step (2) takes $O((|\mathcal{A}| + |\mathcal{Q}|)|E_{\mathcal{Q}}| + (||\mathcal{A}|| + |\mathcal{Q}|)|V_{\mathcal{Q}}|^2)$ time by the complexity of EBChk.

Remark. One might want to find a *minimum* M-extension $\mathcal{A}_M$ of $\mathcal{A}$ such that $\mathcal{Q}$ is instance-bounded under $\mathcal{A}_M$, and $\mathcal{A}_M$ has the least number of access constraints among all M-extensions of $\mathcal{A}$ that make $\mathcal{Q}$ instance-bounded in G, which is formulated as the MBE problem below:

- Input: access schema $\mathcal{A}$, query workload $\mathcal{Q}$, M, N
- Output: whether there exists an M-extension $\mathcal{A}_M$ of $\mathcal{A}$ such that $\mathcal{Q}$ is instance-bounded under $\mathcal{A}_M$ and $|\mathcal{A}_M| \leqslant N$.

The optimization version of the problem is to find such minimum $\mathcal{A}_M$.

Unfortunately, it is NP-hard and log APX-hard to find such a minimum M-extension for given $\mathcal{Q}$, $\mathcal{A}$, M and G. Here log APX-hard problems are NP optimization problems for which no PTIME algorithms have approximation ratio below $c \log n$, where c is some constantand n is the input size (cf. [Aus99]).

Proposition 57: *(1) The decision problem of* MBE *is* NP-*complete. (2) The optimization problem of* MBE *is* log APX-*hard.* □

Proof: (1) We first show the NP-completeness, where the reduction for the lower bound is also an AP-reduction for proving (2).

Upper bound. We show it is in NP by giving an NP algorithm working as follows: guess an M-extension $\mathcal{A}_M$ from the set of all possible type (1) or type (2) additional access constraints, and then check whether $\mathcal{Q}$ is instance-bounded under $\mathcal{A}_M$ in G and $|\mathcal{A}_M| \leqslant N$. Return YES if it is and NO if all such M-extensions are checked.

Lower bound. We show it is NP-hard by reduction from the MINIMUM CARDINALITY KEY problem, which is shown NP-complete (cf. [AB96]). An instance of the MINIMUM CARDINALITY KEY problem consists of a set of functional dependencies $F = \{F_1, \ldots, F_n\}$ over a set of attributes $A = \{A_1, \ldots, A_m\}$ and a natural number N. It is to determine whether there exists a key $K \subseteq A$ implied by F with $|K| \leqslant N$.

Given such an instance of MINIMUM CARDINALITY KEY problem, we construct an instance of MBE below, namely, a set of subgraph queries $\mathcal{Q}$, an access schema $\mathcal{A}$, a data graph G, a natural numbers K, such that there exists a dominating set V' in G with $|V'| \leqslant N$ if and only if there exists an M-extension $\mathcal{A}_M$ of $\mathcal{A}$ with $|\mathcal{A}_M| \leqslant K$.

(a) $\mathcal{Q}$. We let $\mathcal{Q}$ consist of only one subgraph query Q, defined below. For each functional dependency $F_i = X \to Y$ in F with $X = A_{i_1}A_{i_2}\ldots A_{i_p}$ and $Y = A_{i^1}\ldots A_{i^q}$, we include

- $|X|$ (*i.e.*, p nodes) $u_{A_{i_1}}, \ldots, u_{A_{i_p}}$ in V_Q labeled with $A_{i_1}, \ldots, A_{i_p}$, to encode attributes in X;

- ◦ a node u_Y labeled with Y in V_Q to encode the set of attributes in Y;
- ◦ $|Y|$ (*i.e.*, q nodes) $u_{A_{i1}}, \ldots, u_{A_{iq}}$ with labels $A_{i^1}, \ldots, A_{i^p}$ in A in V_Q, to encode attributes in Y;
- ◦ $|X|+|Y|$ (*i.e.*, $p+q$ edges) in E_Q: p edges $(u_{A_{i_j}}, u_Y)$ for all $j \in [1,p]$ and q edges $(u_Y, u_{A_{ij}})$ for all $j \in [1,q]$.

(b) $\mathcal{A}$. We include the following access constraints in $\mathcal{A}$ for each functional dependency $F_i = X \to Y$: $X \to (Y,1)$ and $Y \to (A_{i^j},1)$ for each $j \in [1,q]$.

(c) G. G is the same to Q except that there are M more Y-nodes for each functional dependency $X \to Y$ in F. This is to prevent those Y-nodes from being indexed in any M-extension $\mathcal{A}_M$ of $\mathcal{A}$.

(d) K. Let $K = N + |\mathcal{A}|$.

We next show that there exists a key for A with no more than N attributes if and only if there exists an M-extension with no more than K access constraints.

$\boxed{\Rightarrow}$ Assume there is a key S for A such that $|S| \leqslant N$. We construct an M-extension $\mathcal{A}_M$ of $\mathcal{A}$ as follows. For each attribute A in S, add $\emptyset \to (A,1)$ to $\mathcal{A}$. One can see that, by incorporating these $|S|$ new type (1) and type (2) access constraints, $\mathcal{A}_M$ is an M-extension of $\mathcal{A}$, and moreover, Q is instance-bounded under $\mathcal{A}_M$ and $|\mathcal{A}_M| = |\mathcal{A}| + N = K$.

$\boxed{\Leftarrow}$ Suppose there exists an M-extension $\mathcal{A}_M$ of $\mathcal{A}$ such that Q is instance-bounded under $\mathcal{A}_M$ and moreover $|\mathcal{A}_M| - |\mathcal{A}| \leqslant N$. Note that those nodes labeled with Y (set names) cannot be indexed by extra access constraints, and moreover, adding type (2) access constraints will not have any impact on the boundedness of Q. Thus, type (1) access constraints in $\mathcal{A}_M \setminus \mathcal{A}$ actually encoded a key S for the set A of attributes, with $|S| \leqslant N$.

(2) Note that the M-extension $\mathcal{A}_M$ and keys for F have a 1-1 correspondence. That is, the above Karp-reduction actually gives us a AP reduction the optimized version of MINIMUM CARDINALITY KEY problem to MEB. Since the former is log APX-hard [AB96], so is MBE. □

6.5 Boundedly Evaluable Simulation Queries

We have seen that bounded evaluability helps us answer subgraph queries in big graphs within constrained resources. A natural question asks whether the same idea works for simulation queries, which are non-localized and recursive.

This section settles this question in positive. For boundedly evaluable simulation queries, we provide (1) a characterization (Section 6.5.1); (2) a checking algorithm (Section 6.5.2); and (3) an algorithm for generating boundedly evaluable and worst-case optimal query plans (Section 6.5.3), all with the same complexity as their counterparts for subgraph queries. We also give (4) an algorithm for making a finite set of unbounded simulation queries instance-bounded (Section 6.5.4). We contend that the bounded evaluability approach is generic: it works on general pattern queries, localized or non-localized.

6.5.1 Characterization for Simulation Queries

Simulation queries introduce challenges to the analysis.

Example 34: Consider the simulation query $Q_1(V_1, E_2)$ of Example 28, and an access schema $\mathcal{A}_1$ with $\varphi_A = B \to (A,2)$, $\varphi_B = CD \to (B,2)$, $\varphi_C = \emptyset \to (C,1)$, and $\varphi_D = \emptyset \to (D,1)$. One can verify that $\mathsf{VCov}(Q_1, \mathcal{A}_1) = V_1$ and $\mathsf{ECov}(Q_1, \mathcal{A}_1) = E_1$. However, Q_1 is not boundedly evaluable. Indeed, G_1 of Fig. 6.2 matches Q_1, and the maximum match relation $Q_1(G_1)$ "covers" a cycle in G_1 with length proportional to $|G_1|$. That is, while $\mathcal{A}_1$ constrains the neighbors of each node in Q_1, it does not suffice: as shown in Example 28, to check whether v_1 of G_1 matches u_1 of Q_1, we need to inspect nodes of G_1 far beyond the neighbors of v_1, due to the non-localized and recursive nature of simulation queries. □

This suggests a stronger notion of node covers. The *node cover* of an access schema $\mathcal{A}$ on a simulation query Q, denoted by $\mathsf{sVCov}(Q, \mathcal{A})$, is the set of nodes in Q computed as follows:

(a) if a type (1) constraint $\emptyset \to (l, N)$ is in $\mathcal{A}$, then for each node u in Q with label l, $u \in \mathsf{sVCov}(Q, \mathcal{A})$; and

(b) if $S \to (l, N)$ is in $\mathcal{A}$, then for each S-labeled set V_S in Q, a common neighbor u of V_S in Q is in $\mathsf{sVCov}(Q, \mathcal{A})$ if (i) u is labeled with l, (ii) $V_S \subseteq \mathsf{sVCov}(Q, \mathcal{A})$ and (iii) for each node u_S in V_S, (u, u_S) is an edge of Q.

As opposed to VCov for subgraph queries, a node u is in $\mathsf{sVCov}(Q, \mathcal{A})$ if in any graph $G \models \mathcal{A}$, the number of candidate matches of u is bounded in G, *no matter whether* these nodes are in the same neighborhood *or not*. We include u in $\mathsf{sVCov}(Q,\mathcal{A})$ only if some of its children are covered by $\mathcal{A}$ and they bound the candidate matches of u by an access constraint. When we enforce $V_Q = \mathsf{sVCov}(Q,\mathcal{A})$ (see Theorem 35 below), this ensures that *all children* of u have a bounded number of candidates in G. This rules out unbounded matches when retrieving maximum matches by using the indices of $\mathcal{A}$.

The *edge cover* of $\mathcal{A}$ on Q, denoted by $\mathsf{sECov}(Q,\mathcal{A})$, is defined in the same way as $\mathsf{ECov}(Q,\mathcal{A})$ for subgraph queries (Section 6.2), using $\mathsf{sVCov}(Q,\mathcal{A})$ instead of $\mathsf{VCov}(Q,\mathcal{A})$.

Covers for simulation queries are more restrictive than their counterparts for subgraph queries: $\mathsf{sVCov}(Q,\mathcal{A}) \subseteq \mathsf{VCov}(Q,\mathcal{A}) \subseteq V_Q$ and $\mathsf{sECov}(Q,\mathcal{A}) \subseteq \mathsf{ECov}(Q,\mathcal{A}) \subseteq E_Q$.

Analogous to Theorem 51, one can verify the following.

Theorem 58: *A simulation query $Q(V_Q,E_Q)$ is boundedly evaluable under an access schema $\mathcal{A}$ if and only if $V_Q = \mathsf{sVCov}(Q,\mathcal{A})$ and $E_Q = \mathsf{sECov}(Q,\mathcal{A})$.* □

The proof is along the same lines as the proof of Theorem 51, based on a *weak data locality* for simulation queries, analogous to the locality property for subgraph queries.

Weak data locality. *A query Q is* weakly localized *if for any graph G that matches Q, any node u and u' in Q such that (u,u') is an edge in Q, and for any match v of u in G, there must exist a match v' of u' in G such that (v,v') is an edge in G.*

Example 35: Recall Q_1 and $\mathcal{A}_1$ from Example 34. One can verify that neither u_1 nor u_2 in Q_1 is in $\mathsf{sVCov}(Q_1,\mathcal{A}_1)$ and hence, Q_1 is not boundedly evaluable under $\mathcal{A}_1$ by Theorem 58. This is consistent with the observation of Example 34.

Now define $Q_2(V_2,E_2)$ by reversing the directions of (u_3,u_2) and (u_4,u_2) in Q_1. Then $\mathsf{sVCov}(Q_2,\mathcal{A}_1) = V_2$ and $\mathsf{sECov}(Q_2,\mathcal{A}_1) = E_2$. Hence, Q_2 is boundedly evaluable under $\mathcal{A}_1$ by Theorem 58. Given G_1 of Fig. 6.2, we can find $Q_2(G_1) = \emptyset$ without fetching the unbounded cycle of G_1. □

6.5.2 Deciding Boundedly Evaluability of Simulation Queries

We now revisit $\mathsf{EBnd}(Q,\mathcal{A})$ (Section 6.2): given a simulation query Q and an access schema $\mathcal{A}$, it is to decide whether Q is boundedly evaluable under $\mathcal{A}$. We show that graph simulation does not increase the complexity of $\mathsf{EBnd}(Q,\mathcal{A})$.

Theorem 59: *For simulation queries Q,* $\mathsf{EBnd}(Q,\mathcal{A})$ *has the same complexity as for subgraph queries, in both the general case and the two special cases stated in Theorem 52.* □

To prove Theorem 59 we give a checking algorithm, denoted by sEBChk, which is the same as EBChk of Fig. 6.3 except that it uses a revised notion of actualized constraints. For each $S \rightarrow (l,N)$ in $\mathcal{A}$ with $S \neq \emptyset$, and each node u in Q with $f_Q(u) = l$, its *actualized constraint for simulation* is $\bar{V}_S^u \mapsto (u,N)$, where $\bar{V}_S^u$ is the maximum set of neighbors of u in Q such that (a) there exists a S-labeled set $V_S \subseteq \bar{V}_S^u$, and (b) for each $u' \in \bar{V}_S^u$, (i) $f_Q(u') \in S$; and (ii) (u,u') is an edge of Q. In contrast to its counterpart defined in Section 6.2, this notion further requires condition (ii) to cope with $\mathsf{sVCov}(Q,\mathcal{A})$.

Example 36: Given $Q_2(V_2,E_2)$ and $\mathcal{A}_1$ considered in Example 35, sEBChk first computes the set Γ of actualized constraints for $\mathcal{A}_1$ on Q_2: $\phi_1 = (u_3,u_4) \mapsto (u_2,2)$, $\phi_2 = u_2 \mapsto (u_1,2)$. It then initializes both $\mathcal{B}$ and $\mathcal{C}$ to be $\{u_3,\ u_4\}$, sets $ct[\phi_1] = 2$, $ct[\phi_2] = 1$, and initializes lists $L[u_1], \ldots, L[u_4]$ accordingly (see Fig. 6.3). As in Example 31, it finds that $V_2 \subseteq \mathcal{C}$ and that each edge of E_2 is covered by some constraint in $\mathcal{A}_1$. Thus it returns "yes", *i.e.,* Q_2 is boundedly evaluable under $\mathcal{A}_1$. □

The correctness of sEBChk follows from the characterization of Theorem 58. Along the same lines as the analysis of EBChk, the proof uses the following property of $\mathsf{sVCov}(Q,\mathcal{A})$: a node u of Q is in $\mathsf{sVCov}(Q,\mathcal{A})$ if and only if either

- there exists $\emptyset \rightarrow (l,N)$ in $\mathcal{A}$ and $f_Q(u) = l$; or
- $\bar{V}_S^u \mapsto (u,N)$ and there exists a S-labeled set of Q that is a subset of $\bar{V}_S^u \cap \mathsf{sVCov}(Q,\mathcal{A})$.

Algorithm sEBChk has the same complexity as EBChk: sEBChk is the same as EBChk except the computation of the set Γ of all actualized constraints (lines 1-2 of Fig. 6.3), which remains in $O(|\mathcal{A}||E_Q|)$ time, the same as for subgraph queries.

6.5.3 Generating Bounded Query Plans

We next show that for boundedly evaluable simulation queries Q under an access schema $\mathcal{A}$, we can generate query plans $\mathcal{P}$ such that in any graph G, $\mathcal{P}$ computes $Q(G)$ by accessing a bounded subgraph G_Q of Q, leveraging the indices of $\mathcal{A}$, such that $Q(G) = Q(G_Q)$. Indeed, Theorem 54, the result for subgraph queries, carries over to simulation queries.

Theorem 60: *There exists an algorithm that, given any boundedly evaluable simulation query Q under an access schema $\mathcal{A}$, generates an boundedly evaluable and worst-case optimal query plan in $O(|V_Q||E_Q||\mathcal{A}|)$ time.* □

We show that a minor revision sQPlan of algorithm QPlan (Fig. 6.4) suffices to do these, retaining the same complexity as QPlan. The only difference is that we use actualized constraints for simulation given above, and the stronger notion of node covers instead of data locality.

Example 37: Given $Q_2(V_2, E_2)$ of Example 35 and $\mathcal{A}_1$ of Example 34, sQPlan generates a query plan $\mathcal{P}$. Using the set Γ of actualized constraints of $\mathcal{A}_1$ on Q_2 (see Example 36), QPlan builds $Q_\Gamma(V_\Gamma, E_\Gamma)$, where $V_\Gamma = V_2$, and E_Γ contains (u_3, u_2), (u_4, u_2) and (u_2, u_1). Initially, it adds $\mathsf{fetch}(u_3, nil, \varphi_C, true)$ and $\mathsf{fetch}(u_4, nil, \varphi_D, true)$ to $\mathcal{P}$. It then finds that u_2 and u_1 can be deduced from u_3 and u_4 by using Q_Γ, and thus adds $\mathsf{fetch}(u_2, \{u_3, u_4\}, \varphi_B, true)$ and $\mathsf{fetch}(u_1, \{u_2\}, \varphi_A, true)$ to $\mathcal{P}$.

For any graph $G \models \mathcal{A}_1$, we compute $Q_2(G)$ by using $\mathcal{P}$. It retrieves 8 candidate matches for nodes in Q_2, *i.e.,* 4 for u_1, 2 for u_2, 1 for each of u_3 and u_4. It then finds at most 12 edges between these candidates that are possible edge matches by using the indices of $\mathcal{A}_1$: 4 for each of (u_1, u_2) and (u_2, u_1), and 2 for each of (u_2, u_3) and (u_2, u_4). That is, $\mathcal{P}$ fetches a subgraph G_{Q_2} of Q_2, by accessing 8 nodes and 12 edges. □

6.5.4 Making Simulation Queries Instance Bounded

Finally, we study finite sets $\mathcal{Q}$ of simulation queries when they are not boundedly evaluable under an access schema $\mathcal{A}$. We show that Proposition 55 also holds here: for any graph $G \models \mathcal{A}$, there exists an M-bounded extension $\mathcal{A}_M$ of $\mathcal{A}$ under which $\mathcal{Q}$ is instance-bounded in G for some bound M

For a predefined and small M, we revisit $\mathsf{EEP}(\mathcal{Q}, \mathcal{A}, M, G)$ to decide whether there exists an M-bounded extension $\mathcal{A}_M$ of $\mathcal{A}$ that makes $\mathcal{Q}$ instance-bounded in G (see Section 6.4). We show that Theorem 56 remains intact on simulation queries.

Theorem 61: *For simulation queries,* $\mathsf{EEP}(\mathcal{Q}, \mathcal{A}, M, G)$ *is in* $O(|G| + (|\mathcal{A}| + |\mathcal{Q}|)|E_Q| + (||\mathcal{A}|| + |\mathcal{Q}|)|V_Q|^2)$ *time.* □

As a proof, we show that a minor revision sEEChk of EEChk (Section 6.4) can check EEP for simulation queries, with the same complexity as EEChk. Indeed, the algorithm sEEChk works exactly the same to EEChk except that it invokes sEBChk instead of EBChk for checking the bounded evaluability. Thus the complexity follows from EEChk. The correctness follows from EEChk and sEBChk.

6.6 Experimental Study

Using real-life data, we conducted three sets of experiments to evaluate (1) the effectiveness of our query evaluation approach based on bounded evaluability, (2) the effectiveness of instance boundedness and (3) the efficiency of our algorithms.

Experimental setting. We used three real-life datasets.

Internet Movie Data Graph (IMDbG) was generated from the Internet Movie Database (IMDb) [IMD], with 5.1 million nodes, 19.5 million edges and 168 labels in IMDbG.

Knowledge graph (DBpediaG) was taken from DBpedia 3.9 [DBp], with 4.1 million nodes, 19.5 million edges and 1434 labels.

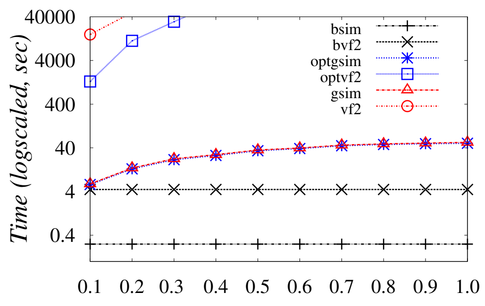
Webbase-2001 (WebBG) recorded Web pages produced in 2001 [Web], in which nodes are URLs, edges are directed links between them, and labels are domain names of the URLs. It has 118 million nodes, 1 billion edges and 0.18 million labels.

Access schema. We extracted 168, 315 and 204 access constraints from IMDbG, DBpediaG and WebBG, respectively, by using degree bounds, label frequencies and data semantics. For example, (actress, year) $\rightarrow$ (feature_film, 104) is a constraint on IMDbG, stating that each actress starred in no more than 104 feature films per year. We found it easy to extract access constraints from real-life data. There are many more constraints for our datasets, which we did not use in our tests.

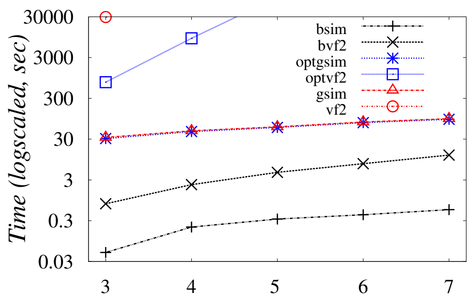
For each constraint $S \rightarrow (l, N)$, we built index by (a) creating a table in which each tuple encodes an actualized constraint $V_S \mapsto (u, N)$; and (b) building an index on the attributes for V_S in the new table, using MySQL 5.5.35.

Pattern queries. For each dataset, we randomly generated 100 pattern queries using its labels, controlled by #n, #e and #p, the number of nodes, edges and match predicates in the ranges [3, 7], [#n-1, 1.5*#n] and [2, 8], respectively. We did not use big patterns to favor conventional methods VF2 and optVF2 (see below), which do not work on large queries.

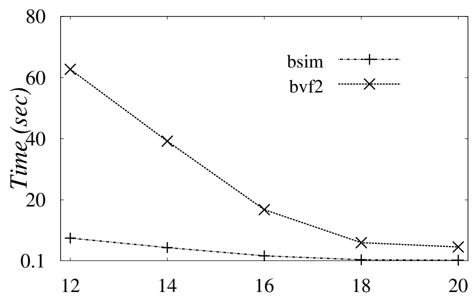
Algorithms. We implemented the following algorithms in C++: (1) EBChk, QPlan, EEChk for subgraph queries, and sEBChk, sQPlan, sEEChk for simulation queries; (2) pattern matching algorithms bVF2 and bSim for subgraph and simulation queries, by using query plans generated by QPlan and sQPlan, respectively; (3) conventional match-



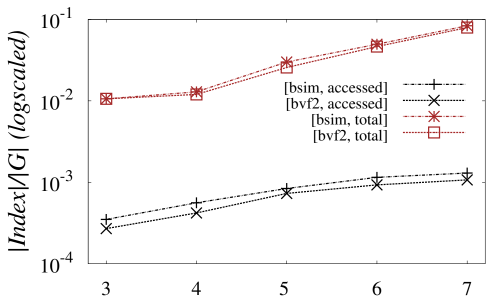
(a) IMDbG: varying $|G|$



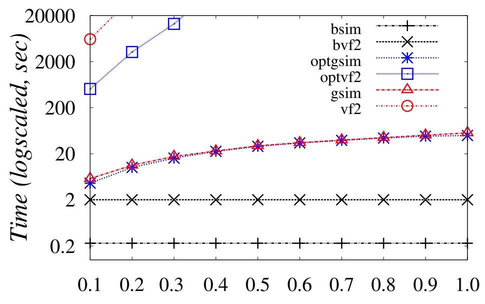
(b) IMDbG: varying Q



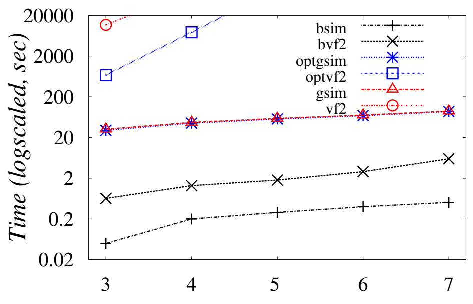
(c) IMDbG: varying $||\mathcal{A}||$



(d) IMDbG: index size



(e) DBpediaG: varying $|G|$



(f) DBpediaG: varying Q

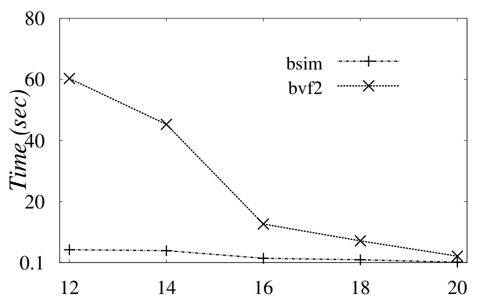
Figure 6.5: Effectiveness of bounded evaluable query evaluation

ing algorithms gsim [HHK95b] and VF2 (using C++ Boost Graph Library) for simulation and subgraph queries, respectively, and their optimized versions optgsim and optVF2 by using indices in the access constraints.

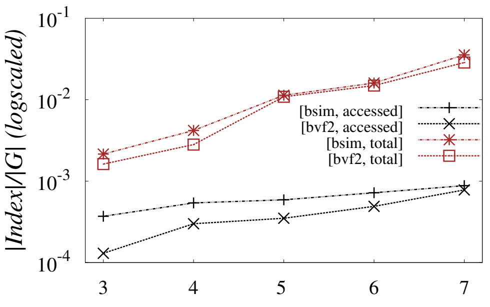
The experiments were conducted on an Amazon EC2 memory optimized instance r3.4xlarge with 122GB memory and 52 EC2 compute units. All the experiments were run 3 times. The average is reported here.

Experimental results. We next report our findings.

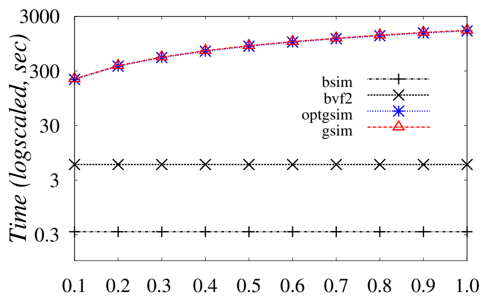
Exp-1: Effectiveness of bounded evaluability.



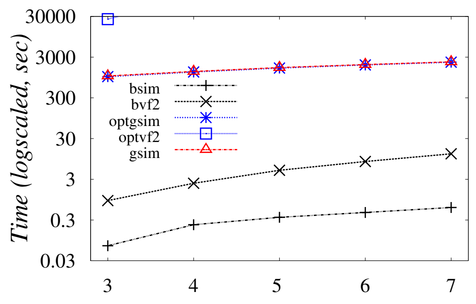
(a) DBpediaG: varying $||\mathcal{A}||$



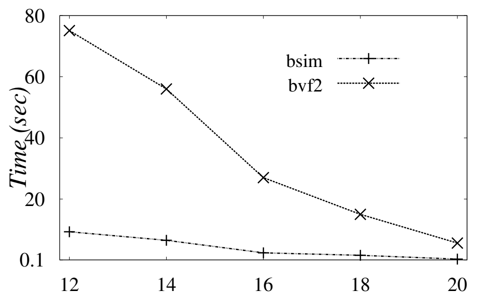
(b) DBpediaG: index size



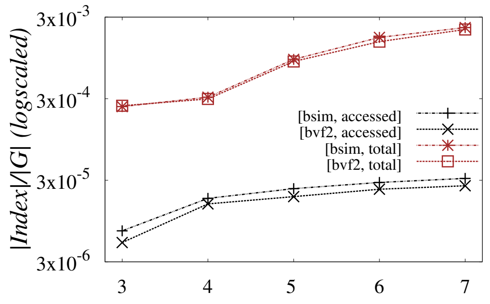
(c) WebBG: varying $|G|$



(d) WebBG: varying Q



(e) WebBG: varying $||\mathcal{A}||$



(f) WebBG: index size

Figure 6.6: Effectiveness of bounded evaluable query evaluation (Cont.)

(1) Percentage of boundedly evaluable queries. We checked the randomly generated queries using algorithms EBChk and sEBChk, and found the following: (1) 61%, 67% and 58% of subgraph queries on IMDbG, DBpediaG and WebBG are boundedly evaluable under the access constraints described above, and (2) 32%, 41% and 33% for simulation queries, respectively. These tell us that (a) by using a small number of simple access constraints, many subgraph and simulation queries are boundedly evaluable; and (b) more subgraph queries are bounded than simulation queries under the same constraints, due to their locality (Section 6.1), as expected.

(2) Effectiveness of bounded queries. To evaluate the impact of boundedly evaluable queries, we compared their running time by bVF2 and bSim (with query plans generated by QPlan and sQPlan) vs. the conventional methods VF2, optVF2 and gsim, optgsim. As VF2 and optVF2 are slow, we only report their performance when they ran to completion. Unless stated otherwise, we used all access constraints and full-size datasets.

(a) Impact of $|G|$. Varying the size $|G|$ by using scale factors from 0.1 to 1, we report the results on the three datasets in Figures 6.5(a), 6.5(e) and 6.6(c). Observe the following. (1) The evaluation time of boundedly evaluable queries is *independent of* $|G|$. Indeed, bVF2 and bSim consistently took 4.45s, 2.02s, 5.8s and 0.25s, 0.23s, 0.34s on all subgraphs of IMDbG, DBpediaG and WebBG, respectively. (2) VF2 and optVF2 could *not* run to completion within 40000s on all subgraphs of WebBG, and on subgraphs of IMDbG and DBpediaG with scale factor above 0.3. On the full-size WebBG, bVF2 took 0.9s as opposed to 25729s by optVF2 for queries that optVF2 could process within reasonable time, at least *28587 times faster*. (3) Algorithms optgsim and gsim are sensitive to $|G|$ (note the logarithmic scale of the y-axis), and are much slower than bSim. For instance, on the full-size WebBG, bSim took 0.34s vs. 1630s by optgsim, *4793 times faster*. The improvement of bVF2 over optVF2 is bigger than that of bSim over optgsim as optVF2 has a higher complexity and thus, is more sensitive to $|G|$.

(b) Impact of Q. To evaluate the impact of patterns, we varied #n of Q from 3 to 7. The results, as shown in Figures 6.5(b), 6.5(f) and 6.6(d), tell us the following. (1) The smaller Q is, the faster all the algorithms are, as expected. (2) For all queries, bVF2 and bSim are efficient: they return answers within 12.7s on all three datasets. (3) Algorithms VF2 and optVF2 do not scale with Q. When #n > 4, none of them could run to completion within 40000s, on all three datasets. (4) Algorithms gsim and optgsim are much slower than bSim for all queries.

(c) Impact of $||\mathcal{A}||$. To evaluate the impact of access constraints on bVF2 and bSim, we varied $||\mathcal{A}||$ from 12 to 20 and processed boundedly evaluable queries using the varied indices in $\mathcal{A}$. As shown in Figures 6.5(c), 6.6(a) and 6.6(e), more access constraints help QPlan and sQPlan get better query plans, as expected. For example, on WebBG, when 20 access constraints were used, bSim and bVF2 took 0.36s and 5.6s, respectively, while they were 9.3s and 75.1s when $||\mathcal{A}|| = 12$.

(3) Size of accessed data. In the same setting as Exp-1(2)(b) above, we examined the size of data accessed by bVF2 and bSim. For each boundedly evaluable query Q, we examined (a) $|\text{accessed}_Q|$, the size of data accessed, and (b) $|\text{index}_Q|$, the size of indices in

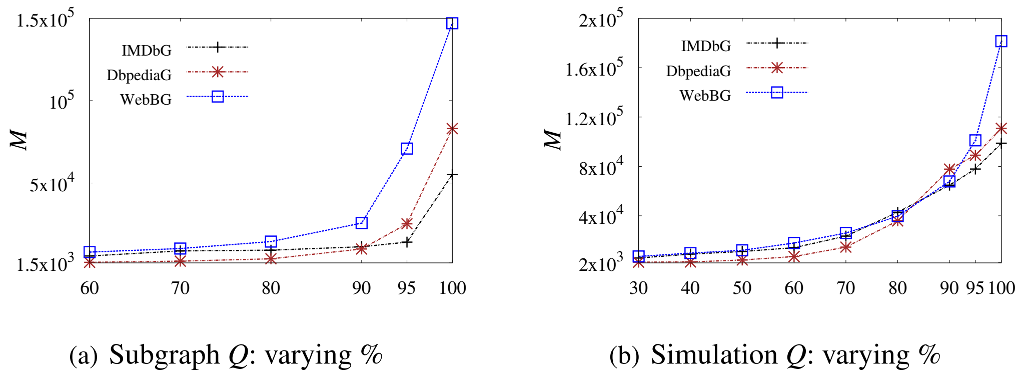


Figure 6.7: Effectiveness of instance boundedness

those access constraints used, by bVF2 and bSim for answering Q. We report the average in Figures 6.5(d), 6.6(b) and 6.6(f). The results tell us that the query plans accessed no more than 0.13% of $|G|$ for all subgraph and simulation queries on all datasets, with indices less than 8% of $|G|$. This further confirms the effectiveness of our approach.

Exp-2: Effectiveness of instance boundedness. Varying x, we examined the minimum M that makes $x\%$ of queries instance-bounded under M-bounded extensions on IMDbG, DBpediaG and WebBG, via EEChk and sEEChk. As Figures 6.7(a) and 6.7(b) show, a small M (compared to $|G|$) suffices to make a large percentage of the queries instance-bounded. For instance, when M is 14113, 25218 and 70916 (resp. 77873, 89068, 101134), over 95% of all subgraph (resp. simulation) queries randomly generated are instance-bounded in IMDbG, DBpediaG and WebBG, respectively; that is, M is 0.057%, 0.107% and 0.006% of $|G|$ (resp. 0.32%, 0.38% and 0.009%). When M is 181448 (0.016% of WebBG), *all* subgraph and simulation queries become instance-bounded in all datasets.

Expt-3: Efficiency. Finally, we evaluated the efficiency of our algorithms. We found that EBChk, QPlan, sEBChk and sQPlan took at most 7ms, 37ms, 6ms and 32ms, respectively, for all queries on three datasets with all the access constraints.

Summary. From the experiments we find the following. (1) The approach by bounded evaluability is practical for pattern queries on large graphs. (a) It is easy to find access constraints from real-life datasets. (b) About 60% (resp. 33%) subgraph (resp. simulation) queries are boundedly evaluable under a small number of access constraints. (c) boundedly evaluable queries scale well with big graphs: their evaluation time is *independent of* $|G|$. (2) The approach is effective for both localized and non-localized queries: bVF2 and bSim outperform optVF2 and optgsim by 4 and 3 orders of magnitude on average on WebBG, respectively. (3) A small M suffices to make queries instance-bounded: 0.006% (resp. 0.009%) of $|G|$ for 95% of subgraph (resp. simulation) queries on WebBG, and 0.013% (resp. 0.016%) to bound *all* queries. (4) Our algorithms are efficient: they take no more than 37ms in all cases.

Summary

We propose to answer graph pattern queries in big graphs by making use of bounded evaluability. We have developed techniques underlying the approach: access constraints on graphs, boundedly evaluable pattern queries, characterizations and algorithms for deciding whether pattern queries are bounded, algorithms for generating (worst-case) optimal query plans if so, and otherwise, algorithms for making queries instance-bounded. We have verified, analytically and experimentally, the effectiveness of the approach: it works for both localized queries and non-localized queries.

Part IV

Conclusion

Chapter 7

Conclusion and Future Work

This chapter summarizes the results of this dissertation and proposes further research directions.

7.1 Summary

This dissertation proposes bounded evaluability to query big data, relational or graph, by accessing a bounded amount of data, to compute exact answers if possible with or without views, or approximate answers with accuracy bounds otherwise. The idea is to leverage indices built according to cardinality constraints conformed by the datasets of concern. More specifically, the contribution in the work includes the following.

- We have formalized bounded evaluability and studied the complexity of bounded evaluability analysis for several fragments of FO (Chapter 2).
- We have developed effective syntax of boundedly evaluable queries to handle the undecidability of bounded evaluability analysis, and to make full practical use of the idea (Chapter 3).
- We have developed and implemented a bounded evaluation framework on top of DBMS with practical algorithms, based on the effective syntax (Chapter 3).
- We have developed a bounded-resource approximation scheme that extends bounded evaluability to bounded approximation, under an extended access schema (Chapter 4).
- We have incorporated materialized views with bounded evaluability, investigated the possibility of bounded rewriting using views, and developed effective syntax for queries that have a bounded rewriting (Chapter 5).

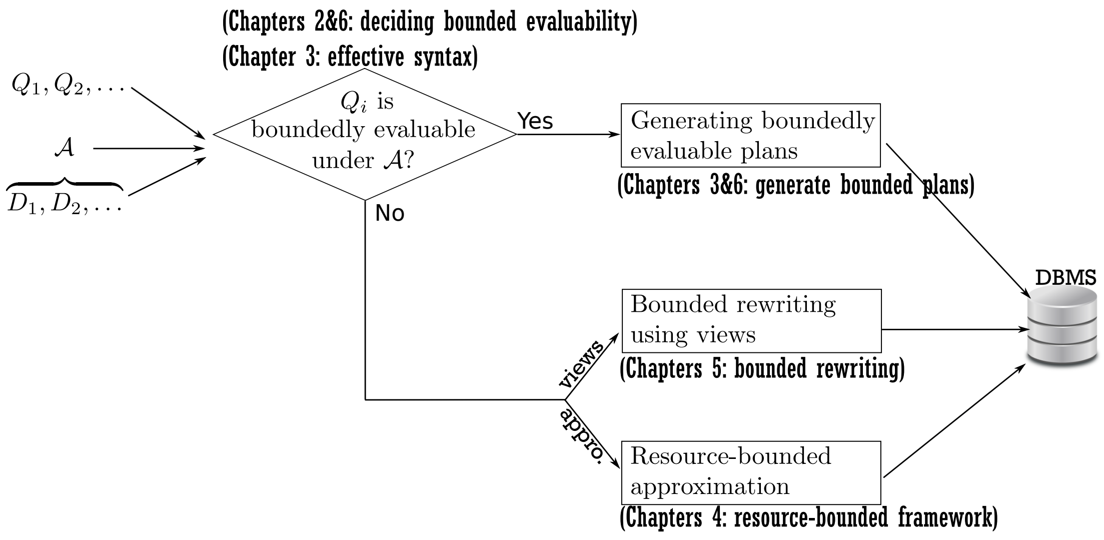

(Chapters 2&6: deciding bounded evaluability)
(Chapter 3: effective syntax)
$Q_1, Q_2, \ldots$
$\mathcal{A}$
$D_1, D_2, \ldots$
Q_i is boundedly evaluable under $\mathcal{A}$?
Yes
No
Generating boundedly evaluable plans
(Chapters 3&6: generate bounded plans)
DBMS
Bounded rewriting using views
(Chapters 5: bounded rewriting)
views
appro.
Resource-bounded approximation
(Chapters 4: resource-bounded framework)

Figure 7.1: A framework of querying big data with bounded data access

- We have extended bounded evaluability from relational queries to graph pattern queries (Chapter 6).

These results form a framework for *querying big data with bounded data access* on top of existing DBMS, depicted in Fig 7.1. It maintains an access schema $\mathcal{A}$ such that, for any online query Q,

(1) it first checks whether Q is boundedly evaluable under $\mathcal{A}$, via bounded evaluability analysis or the effective syntax (Chapters 2, 3 and 6);

(2) if so, it generates a boundedly evaluable query plan for Q such that, for any big database D (resp. big data graph G) satisfying $\mathcal{A}$, the plan can find exact answers to Q in D (resp. G) using DBMS, by accessing a bounded amount of data whose size is independent of big data D (resp. G) via DBMS while using as few constraints in $\mathcal{A}$ as possible (Chapter 3);

(3) otherwise, it utilizes materialized views and generates a bounded rewriting using the views under $\mathcal{A}$ if possible, to compute exact answers to Q by accessing a bounded fraction of D, besides the views (Chpater 5);

(4) otherwise, it computes approximate answers S to Q in D under $\mathcal{A}$ along with an accuracy bound η for S, by accessing an α-fraction of D, where α can be specified by the user (Chapter 4);

Based on the experimental study, we find that the framework works well in coping with big data. In several real-life datasets, under a few hundred access constraints, 67% RA queries (whose attributes are covered by the constrainst) are boundedly evaluable;

their query plans outperform commercial DBMS by 3 orders of magnitude, and the gap gets larger on bigger D. For queries that are not boundedly evaluable, we can compute approximate answers with bounded accuracy, using resource ratio as small as 5.5×10^{-4} bound (see Chapters 3 and 4). The approach is found effective as well for graph pattern queries with real-life big web-graphs and social networks (see Chapter 6)

7.2 Further Research

This work raises many new research issues. Below are some relevant ideas of interest.

(1) Regarding the analysis of bounded evaluability, an open problem is to find a matching upper and lower bound for $\mathsf{BEP}(\mathsf{CQ})$. While we have shown that $\mathsf{BEP}(\mathsf{CQ})$ is decidable in 2EXPSPACE, it does not match the EXPSPACE-hard lower bound. A matching bound for $\mathsf{BEP}(\mathsf{CQ})$ will also give us a matching bound for BEP (UCQ) and BEP ($\exists\mathsf{FO}^+$). Another interesting problem is to investigate the complexity of BEP when plans are not restricted to those algebra operations occurring in Q, *i.e.,*, for $\mathsf{BEP}(\mathsf{UCQ})$, unions are allowed in the plans even when the queries are CQ. The conjecture is that both $\mathsf{BEP}(\mathsf{UCQ})$ and BEP ($\exists\mathsf{FO}^+$) will be Π_2^p-complete in this setting. Similarly, an interesting extension is to investigate BEP under the setting that plans and queries are defined in different query classes. The fourth topic is to study M-bounded evaluability, where the bounded plans are guaranteed to access no more than M tuples in all databases satisfying the access constraints. This will give the controllability on the maximum cost of bounded query plans. The fifth topic is to investigate bounded evaluability for group-by aggregate queries. An extension of access constraints may be needed to handle the bag semantics in aggregate functions, *e.g.,*sum and count.

(2) Regarding bounded approximation, a topic for future work is to study a bi-criteria optimization problem, to minimize the indexing cost of access schema and to maximize accuracy bound. Another topic is to study bounded approximation in the presence of views. This requires a more sophisticated mechanism to embed views in the generation of bounded query plans and α-bounded execution plans, for a balanced trade-off between data access and accuracy.

(3) Regarding bounded evaluability with views, a topic is to study bounded view maintenance, to incrementally maintain $\mathcal{V}(D)$ by accessing a bounded amount of data in D, in response to changes to D. Another topic is to select views and identify access con-

straints, to maximize queries in an application that have a bounded rewriting. The third topic is to study bounded rewriting using views when views are queries are defined in different query classes.

(4) Regarding the computational model for bounded evaluability, a topic is to study *incremental bounded evaluability*: given an access schema $\mathcal{A}$, a database $D \models \mathcal{A}$ and a query Q, it is to incrementally compute $Q(D \oplus \Delta D)$ in response to all changes ΔD to D, by accessing a bounded amount of data from D under $\mathcal{A}$. Similarly for bounded evaluability on graphs. Another topic is to study bounded evaluability in the distributed setting, where we focus on the feasibility of evaluating queries with bounded communication cost.

(5) Finally, regarding access constraints, a topic is to develop a systematic method for discovering constraints on both relational and graph data. Another topic is to develop algorithms for discovering a (minimum) set of access constraints to cover a workload.

Bibliography

[AB96] Tatsuya Akutsu and Feng Bao. Approximating minimum keys and optimal substructure screens. In *COCOON*, 1996.

[ACG$^+$99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation,Combinatorial optimization problems and their approximability properties*. Springer, 1999.

[ACK$^+$11] Michael Armbrust, Kristal Curtis, Tim Kraska, Armando Fox, Michael J. Franklin, and David A. Patterson. PIQL: Success-tolerant query processing in the cloud. *PVLDB*, 5(3), 2011.

[AFF01] Giorgio Ausiello, Paolo Giulio Franciosa, and Daniele Frigioni. Directed hypergraphs: Problems, algorithmic results, and a novel decremental approach. In *ICTCS*, 2001.

[AFP$^+$09] Michael Armbrust, Armando Fox, David A. Patterson, Nick Lanham, Beth Trushkowsky, Jesse Trutna, and Haruki Oh. SCADS: Scale-independent storage for social computing applications. In *CIDR*, 2009.

[Afr11] Foto N. Afrati. Determinacy and query rewriting for conjunctive queries and views. *TCS*, 412(11):1005–1021, 2011.

[AGPR99] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, 1999.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[ALK$^+$13] Michael Armbrust, Eric Liang, Tim Kraska, Armando Fox, Michael J. Franklin, and David Patterson. Generalized scale independence through incremental precomputation. In *SIGMOD*, 2013.

[ALU07] Foto N. Afrati, Chen Li, and Jeffrey D. Ullman. Using views to generate efficient evaluation plans for queries. *JCSS*, 73(5):703–724, 2007.

[AMH08] Daniel J. Abadi, Samuel Madden, and Nabil Hachem. Column-stores vs. row-stores: how different are they really? In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, 2008.

[AMP$^+$13] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. BlinkDB: Queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.

[Aus99] Giorgio Ausiello. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer, 1999.

[BBB13] Vince Bárány, Michael Benedikt, and Pierre Bourhis. Access patterns and integrity constraints revisited. In *ICDT*, 2013.

[BCD03a] Brian Babcock, Surajit Chaudhuri, and Gautam Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, 2003.

[BCD03b] Brian Babcock, Surajit Chaudhuri, and Gautam Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, 2003.

[BHK$^+$10] Joel Brynielsson, Johanna Högberg, Lisa Kaati, Christian Martenson, and Pontus Svenson. Detecting social positions using simulation. In *ASONAM*, 2010.

[BRL13] Pablo Barceló, Juan L. Reutter, and Leonid Libkin. Parameterized regular expressions and their languages. *TCS*, 474:21–45, 2013.

[BTSa] BTS. *http://www.transtats.bts.gov/DatabaseInfo.asp?DB_ID=120.*

[BTSb] BTS. *http://www.transtats.bts.gov/DatabaseInfo.asp?DB_ID=110.*

[CCC$^+$98] Moses Charikar, Chandra Chekuri, Toyat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. In *SODA*, 1998.

[CF16] Yang Cao and Wenfei Fan. An effective syntax for bounded relational queries. In *SIGMOD*, 2016.

[CFGL16] Yang Cao, Wenfei Fan, Floris Geerts, and Ping Lu. Bounded query rewriting using views. In *PODS*, 2016.

[CFH15] Yang Cao, Wenfei Fan, and Ruizhe Huang. Making pattern queries bounded in big graphs. In *ICDE*, 2015.

[CFY14] Yang Cao, Wenfei Fan, and Wenyuan Yu. Bounded conjunctive queries. *PVLDB*, 2014.

[CG05] Graham Cormode and Minos Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, 2005.

[CGHJ12] Graham Cormode, Minos Garofalakis, Peter J Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *FTDB*, 4(1–3), 2012.

[CGN15] Boris Cule, Floris Geerts, and Reuben Ndindi. Space-bounded query approximation. In *ADBIS*, 2015.

[CGRS01] Kaushik Chakrabarti, Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Approximate query processing using wavelets. *VLDB J.*, 10(2-3), 2001.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, 1977.

[CNS99] Sara Cohen, Werner Nutt, and Alexander Serebrenik. Rewriting aggregate queries using views. In *PODS*, 1999.

[DBp] DBpedia. Dbpedia. *http://wiki.dbpedia.org/Downloads39*.

[DLN07] Alin Deutsch, Bertram Ludäscher, and Alan Nash. Rewriting queries using views with access patterns under integrity constraints. *TCS*, 371(3), 2007.

[Dob05] Alin Dobra. Histograms revisited: When are histograms the best approximation method for aggregates over joins. In *PODS*, 2005.

[Elm08] Ramez Elmasri. *Fundamentals of database systems*. Pearson Education India, 2008.

[Faca] Facebook. *https://en-gb.facebook.com/about/graphsearch*.

[Facb] *http://newsroom.fb.com.*

[FE13] Robert W. Faris and Susan T. Ennett. Adolescent aggression: The role of peer group status motives, peer aggression, and group characteristics. *Social Networks*, 34(4), 2013.

[FGC$^+$15] Wenfei Fan, Floris Geerts, Yang Cao, Ting Deng, and Ping Lu. Querying big data by accessing small data. In *PODS*, 2015.

[FGL14] Wenfei Fan, Floris Geerts, and Leonid Libkin. On scale independence for querying big data. In *PODS*, 2014.

[FGN13] Wenfei Fan, Floris Geerts, and Frank Neven. Making queries tractable on big data with preprocessing. *PVLDB*, 2013.

[GBDS14] Ivana Grujic, Sanja Bogdanovic-Dinic, and Leonid Stoimenov. Collecting and analyzing data from e-government facebook pages. In *ICT Innovations*, 2014.

[GJ79] Michael Garey and David Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[GN14] Andrey Gubichev and Thomas Neumann. Exploiting the query structure for efficient join ordering in SPARQL queries. In *EDBT*, 2014.

[Gova] UK Government. *http://data.gov.uk/dataset/road-accidents-safety-data.*

[Govb] UK Government. *http://data.gov.uk/dataset/naptan.*

[GT91] Allen Van Gelder and Rodney W. Topor. Safety and translation of relational calculus queries. *TODS*, 16(2), 1991.

[Hal01] Alon Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4), 2001.

[HHK95a] M. R. Henzinger, T. Henzinger, and P. Kopke. Computing simlations on finite and infinite graphs. In *FOCS*, 1995.

[HHK95b] Monika Rauch Henzinger, Thomas A Henzinger, and Peter W Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, 1995.

[HHW97] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online aggregation. In *SIGMOD*, 1997.

[HKPT99] Y Huhtala, Juha Kärkk ainen, Pasi Porkka, and Hannu Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[ICDK14] Ioana Ileana, Bogdan Cautis, Alin Deutsch, and Yannis Katsis. Complete yet practical search for minimal query reformulations under constraints. In *SIGMOD*, pages 1015–1026, 2014.

[IMD] IMDb. *http://www.imdb.com/stats/search/* .

[IP99] Yannis E. Ioannidis and Viswanath Poosala. Histogram-based approximation of set-valued query-answers. In *VLDB*, 1999.

[JKM$^+$09] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. In *VLDB*, 2009.

[Klu88] A. Klug. On conjunctive queries containing inequalities. *J. ACM*, 35(1):146–160, 1988.

[KMT98] Phokion G. Kolaitis, David L. Martin, and Madhukar N. Thakur. On the complexity of the containment problem for conjunctive queries with built-in predicates. In *PODS*, 1998.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.

[Lev] Charles Levine. TPC benchmarks. *http://research.microsoft.com /en-us/um/people/gray/WICS_99_TP/22_Levine.ppt.*

[Li03] Chen Li. Computing complete answers to queries in the presence of limited access patterns. *VLDB J.*, 12(3), 2003.

[LLLC12] Jixue Liu, Jiuyong Li, Chengfei Liu, and Yongfeng Chen. Discover dependencies from data - A review. *TKDE*, 24(2), 2012.

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *PODS*, 1995.

[MFE13] Guido Moerkotte, Pit Fender, and Marius Eich. On the correct and complete enumeration of the core search space. In *SIGMOD*, 2013.

[NL04] Alan Nash and Bertram Ludäscher. Processing first-order queries under limited access patterns. In *PODS*, 2004.

[NSV10] Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *TODS*, 35(3), 2010.

[ORK$^+$13] Kirk Ogaard, Heather Roy, Sue Kase, Rakesh Nagi, Kedar Sambhoos, and Moises Sudit. Discovering patterns in social networks with graph matching algorithms. In *SBP*, 2013.

[Pap94] Christos H Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Pop01] Lucian Popa. Object/relational query optimization with chase and backchase. *IRCS Technical Reports Series*, 2001.

[Pos] Postgresql. *https://wiki.postgresql.org/wiki/Index-only_scans*.

[RG00] Raghu Ramakrishnan and Johannes Gehrke. *Database management systems*. McGraw Hill, 2000.

[RR96] G. Ramalingam and Thomas Reps. On the computational complexity of dynamic graph problems. *TCS*, 158(1-2), 1996.

[RSU95] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *PODS*, 1995.

[Smi14] Craig Smith. By the numbers: 98 amazing facebook statistics. DMR, March 2014.

[ST95] Alexei P. Stolboushkin and Michael A. Taitslin. Finite queries do not have effective syntax. In *PODS*, 1995.

[Sto61] Robert R. Stoll. Set theory and logic. *W. H. Freeman and Co., San Francisco, Calif.-London*, 1961.

[Sto76] Larry J. Stockmeyer. The polynomial-time hierarchy. *TCS*, 3(1):1–22, 1976.

[SY80] Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, 1980.

[TPC] TPCH. TPCH. *http://www.tpc.org/tpch/*.

[Ull76] Julian R Ullmann. An algorithm for subgraph isomorphism. *JACM*, 23(1), 1976.

[Ull82] Jeffrey D. Ullman. *Principles of Database Systems, 2nd Edition*. Computer Science Press, 1982.

[Var81] Moshe Y. Vardi. The decision problem for database dependencies. *Inf. Process. Lett.*, 12(5), 1981.

[vdM97] Ron van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *JCSS*, 54(1), 1997.

[Web] Webbase. *http://law.di.unimi.it/webdata/webbase-2001/*.