Direct Access for Answers to Conjunctive Queries with Aggregation

Idan Eldar, Nofar Carmeli, Benny Kimelfeld

Direct Access for Answers to Conjunctive Queries with Aggregation

Idan Eldar
Technion – Israel Institute of Technology, Haifa, Israel

Nofar Carmeli
Inria, LIRMM, Univ Montpellier, CNRS, France

Benny Kimelfeld
Technion – Israel Institute of Technology, Haifa, Israel

Abstract

We study the fine-grained complexity of conjunctive queries with grouping and aggregation. For some common aggregate functions (e.g., min, max, count, sum), such a query can be phrased as an ordinary conjunctive query over a database annotated with a suitable commutative semiring. Specifically, we investigate the ability to evaluate such queries by constructing in log-linear time a data structure that provides logarithmic-time direct access to the answers ordered by a given lexicographic order. This task is nontrivial since the number of answers might be larger than log-linear in the size of the input, and so, the data structure needs to provide a compact representation of the space of answers.

In the absence of aggregation and annotation, past research provides a sufficient tractability condition on queries and orders. For queries without self-joins, this condition is not just sufficient, but also necessary (under conventional lower-bound assumptions in fine-grained complexity). We show that all past results continue to hold for annotated databases, assuming that the annotation itself is not part of the lexicographic order. On the other hand, we show infeasibility for the case of count-distinct that does not have any efficient representation as a commutative semiring. We then investigate the ability to include the aggregate and annotation outcome in the lexicographic order. Among the hardness results, standing out as tractable is the case of a semiring with an idempotent addition, such as those of min and max. Notably, this case captures also count-distinct over a logarithmic-size domain.

2012 ACM Subject Classification Theory of computation → Database query languages (principles); Theory of computation → Database query processing and optimization (theory)

Keywords and phrases aggregate queries, conjunctive queries, provenance semirings, commutative semirings, annotated databases, direct access, ranking function, answer orderings, query classification

Digital Object Identifier 10.4230/LIPIcs.ICDT.2024.4

Related Version *Full Version*: `https://arxiv.org/abs/2303.05327`

1 Introduction

Consider a query Q that may have a large number of answers, say cubic in the number of tuples of the input database D. By answering Q via *direct access*, we avoid the materialization of the list of answers, and instead, construct a compact data structure S that supports random access: given an index i, retrieve the ith answer. Hence, direct access evaluation for a query Q consists of two algorithms, one for the structure construction (with the input D), called *preprocessing*, and one for fast access to the answers (with the input S and i). This task is nontrivial when S is considerably cheaper to construct than $Q(D)$. Similarly to past work on direct access [6], we adopt the tractability requirement of linear or quasi-linear time to construct S, and logarithmic time per access. Hence, up to a poly-logarithmic factor, the required construction time is what it takes to read the database (i.e., linear time), and the access time is constant. The structure S can be viewed as a compact representation of $Q(D)$,

in the general sense of Factorized Databases [17], since its size is necessarily quasi-linear and it provides fast access.

Direct access solutions have been devised for Conjunctive Queries (CQs), first as a way to establish algorithms for enumerating the answers with linear preprocessing time and constant delay [4] (and FO queries with restrictions on the database [2]); the preprocessing phase constructs S, and the enumeration phase retrieves the answers by accessing S with increasing indices i. Later, direct access had a more crucial role within the task of enumerating the answers in a uniformly random order [7]. As a notion of query evaluation, direct access is interesting in its own right, since we can view S itself as the "result" of the query in the case where array-like access is sufficient for downstream processing (e.g., to produce a sample of answers, to return answers by pages, to answer q-quantile queries, etc.). But then S has the benefit that it is considerably smaller and faster to produce than the materialized set of answers. Indeed, recent work has studied the complexity of direct access independently (regardless of any enumeration context) [5], and specifically studied which *orders* over the answers allow for such evaluation [6]. In this paper, we continue with the line of work by Carmeli et al. [6] and investigate the ability to support query evaluation via direct access for *aggregate queries*, while focusing on lexicographic orderings of answers.

For illustration, consider the following example, inspired by the FIFA World Cup. Suppose that we have a database of players of teams (countries), sponsors of teams, and goals scored in different games. Specifically, we have three relations: $\textsc{Teams}(\mathit{player}, \mathit{country})$, $\textsc{Sponsors}(\mathit{org}, \mathit{country})$, and $\textsc{Goals}(\mathit{game}, \mathit{player}, \mathit{time})$. The following CQ finds times when sponsors got exposure due to goals of supported teams:

$$Q_1(c, o, p, t) :- \textsc{Teams}(p, c), \textsc{Sponsors}(o, c), \textsc{Goals}(g, p, t)$$

Suppose also that we would like the answers to be ordered lexicographically by their order in the head: first by c (country), then by o (organization), then by p (player), and lastly by t (time). Note that o, c, p and t are *free* variables and g is an *existential* variable. Carmeli et al. [6] studied the ability to evaluate such ordered queries with direct access. In the case of Q_1, the results of Carmeli et al. show that there is an efficient direct access evaluation (since the query is free-connex and there is no "disruptive trio").

Now, suppose that we would like to count the goals per sponsorship and player. In standard CQ notation (e.g., Cohen et al. [9]), we can phrase this query as follows.

$$Q_2(c, o, p, \mathsf{Count}()) :- \textsc{Teams}(p, c), \textsc{Sponsors}(o, c), \textsc{Goals}(g, p, t)$$

Here, the variables p, c, and o are treated as the *grouping variables* (rather than free variables), where each combination of values defines a group of tuples over (c, o, p, g, t) and $\mathsf{Count}()$ simply counts the tuples in the group. Again, we would like to answer this query via direct access. This introduces two challenges. The first challenge is *aggregate construction*: when we access a tuple using S, the aggregate value should be quickly produced as well. The second challenge is *ordering by aggregation*: how can we incorporate the aggregation in the lexicographic order of the answers if so desired by the query? As an example, we may wish to order the answers first by c, then by $\mathsf{Count}()$, and then by o and p; in this case, we would phrase the head accordingly as $Q_2(c, \mathsf{Count}(), o, p)$.

As previously done in the context of algorithms for aggregate queries [15, 20], we also study a semiring alternative to the above formalism of aggregate queries. Specifically, we can adopt the well-known framework of *provenance semiring* of Green, Karvounarakis and Tannen [13] and phrase the query as an ordinary CQ with the annotation carrying the aggregate value (e.g., the number of goals in our example). To reason about random-access

evaluation, we found it more elegant, general, and insightful to support CQs over annotated databases rather than SQL-like aggregate functions. For illustration, we can phrase the above aggregate query Q_2 as the following CQ Q_3, but for a database that is annotated using a specific commutative semiring.

$$Q_3(c, o, p) :- \textsc{Teams}(p, c), \textsc{Sponsors}(o, c), \textsc{Goals}(g, p, t)$$

In a nutshell (the formal definition is in Section 2), the idea is that each tuple is annotated with an element of the semiring, the annotation of each tuple in the group is the product of the participating tuple annotations, and the annotation of the whole group is the sum of all tuple annotations in the group's tuples. In the case of our example with Q_3, we use the numeric semiring $(\mathbb{Z}, +, \cdot, 0, 1)$, and each tuple is annotated simply with the number 1. We can use different semirings and annotations to compute different aggregate functions like sum, min, and max. Here again, we have challenges analogous to the aggregate case: *annotation construction* and *ordering by annotation*. The previous example becomes ordering by c, then by the annotation, and then by o and p. Notationally, we specify the annotation position by the symbol $\star$ and phrase the query as $Q_3(c, \star, o, p) :- \textsc{Teams}(p, c), \textsc{Sponsors}(o, c), \textsc{Goals}(g, p, t)$. We refer to such a query as a CQ*.

In this paper, we study queries in both formalisms—CQs enhanced with aggregate functions and ordinary CQ*s over annotated databases. We usually devise algorithms and upper bounds on general commutative semirings (possibly with additional conditions), as positive results carry over to the aggregate formalism, and we prove cases of specific intractable queries with specific aggregate functions over ordinary (non-annotated) databases.

Our analysis is done in two parts. In Section 4, we study the case where the annotation or aggregation is *not* a part of the lexicographic order; we show that under reasonable assumptions about the complexity of the semiring operations, all results for ordinary databases [6] continue to hold in the presence of annotation (that is, we can solve annotation construction). We conclude the analogous tractability for the common aggregate functions (count, sum, min, max, average), with the exception of count-distinct which cannot be expressed efficiently as a semiring annotation. In Section 5, we study the ability to include the annotation or aggregation in nontrivial positions within the lexicographic order; we show that the picture is more involved there since we hit hardness very quickly, and we need to carefully state the conditions that allow for efficient direct access for important cases.

The remainder of the paper is organized as follows. After preliminary concepts and notation in Section 2, Section 3 defines the challenge of direct access with an underlying order and recalls the state of affairs for ordinary CQs over ordinary databases. We present our analysis in Sections 4 and 5 (as explained in the previous paragraph), and conclude in Section 6. Missing proofs can be found in the full version of the paper [10].

2 Preliminaries

We begin with preliminary notation and terminology that we use throughout the paper.

Databases and conjunctive queries. A *schema* $\mathbf{S}$ is a finite set $\{R_1, \ldots, R_k\}$ of relation symbols. Each relation symbol R is associated with an arity $\mathtt{ar}(R)$, which is a natural number. We assume a countably infinite set Const of *constants* that appear as values of databases. A *database* D over a schema $\mathbf{S}$ maps every relation symbol R of $\mathbf{S}$ to a finite relation $R^D \subseteq \mathsf{Const}^{\mathtt{ar}(R)}$. If $(c_1, \ldots, c_k)$ is a tuple of R^D (where $k = \mathtt{ar}(R)$), then we call the expression $R(c_1, \ldots, c_k)$ a *fact* of D.

A *Conjunctive Query (CQ)* over the schema $\mathbf{S}$ has the form $Q(\vec{x}) :- \varphi_1(\vec{x},\vec{y}), \ldots, \varphi_\ell(\vec{x},\vec{y})$ where $\vec{x}$ and $\vec{y}$ are disjoint sequences of variables, and each $\varphi_i(\vec{x},\vec{y})$ is an *atomic query* $R(z_1,\ldots,z_k)$ such that $R \in \mathbf{S}$ with $\mathtt{ar}(R) = k$ and each z_i is a variable in $\vec{x}$ or $\vec{y}$. Each $\varphi_i(\vec{x},\vec{y})$ is an *atom* of Q, and we denote by atoms(Q) the set of atoms of Q. We call $Q(\vec{x})$ the *head* of the query and $\varphi_1(\vec{x},\vec{y}), \ldots, \varphi_\ell(\vec{x},\vec{y})$ the *body* of the query. The variables of $\vec{x}$ are the *free* variables of Q, and those of $\vec{y}$ are the *existential* variables of Q, and every variable occurs at least once in the body. We use vars(Q) and free(Q) to denote the set of all variables and all free variables of Q, respectively. If $\varphi \in$ atoms(Q), then vars(φ) is the set of variables in φ. We say that Q is *full* if it has no existential variables, that is vars$(Q) =$ free(Q).

We refer to a database D over the schema $\mathbf{S}$ of the CQ Q as a *database over* Q. A *homomorphism* from a CQ Q to a database D over Q is a mapping h from the variables of Q into values of D such that for each atom $R(z_1,\ldots,z_k)$ of Q it holds that $R(h(z_1),\ldots,h(z_k))$ is a fact of D. We denote by $\mathsf{Hom}(Q,D)$ the set of all homomorphisms from Q to D. If $h \in \mathsf{Hom}(Q,D)$ then we denote by $h(\vec{x})$ the tuple obtained by replacing every variable x with the constant $h(x)$, and we denote by $h(\varphi_i(\vec{x},\vec{y}))$ the fact that is obtained from the atom $\varphi_i(\vec{x},\vec{y})$ by replacing every variable z with the constant $h(z)$. An *answer* to Q over D is a tuple of the form $h(\vec{x})$ where $h \in \mathsf{Hom}(Q,D)$. The *result* of Q over D, denoted $Q(D)$, is $Q(D) := \{h(\vec{x}) \mid h \in \mathsf{Hom}(Q,D)\}$.

Consider a CQ $Q(\vec{x}) :- \varphi_1(\vec{x},\vec{y}), \ldots, \varphi_\ell(\vec{x},\vec{y})$. We may refer to Q as $Q(\vec{x})$ to specify the sequence of free variables in the head. In this work, the *order* of the sequence $\vec{x}$ has a crucial role, since it determines the desired order of answers. Specifically, we will assume that the desired order of answers is *lexicographic* in the left-to-right order of $\vec{x}$. For example, the CQ $Q(x_1,x_2) :- R(x_1,x_2), S(x_2,y)$ differs from the CQ $Q'(x_2,x_1) :- R(x_1,x_2), S(x_2,y)$ not only in the order of values within each answer tuple but also in the order over the answers. For $Q(x_1,x_2)$ we order the answers first by x_1 and then by x_2, and for $Q'(x_2,x_1)$ we order first by x_2 and then by x_1,

As usual, we associate a CQ Q with the hypergraph $H(Q) = (V_Q, E_Q)$ where $V_Q =$ vars(Q) and $E_Q = \{\text{vars}(\varphi) | \varphi \in \text{atoms}(Q)\}$. We say that Q is *acyclic* if $H(Q)$ is an (α-)acyclic hypergraph. Recall that a hypergraph $H = (V,E)$ is acyclic if there is a tree $T = (E, E_T)$, called a *join tree* of H, with the running intersection property: for each vertex $v \in V$, the set of hyperedges that contain v induces a connected subtree of T. If H is acyclic and $S \subseteq V$, then we say that H is *S-connex* if H remains acyclic even if we add S to the set of hyperedges [4]. An acyclic CQ Q is *free-connex* if $H(Q)$ is acyclic and free(Q)-connex.

A hypergraph $H' = (V,E')$ is an *inclusive extension* of a hypergraph $H = (V,E)$ if $E \subseteq E'$ and for every edge $e' \in E'$ there is an edge $e \in E$ such that $e' \subseteq e$. It is known that H is acyclic S-connex if and only if H has an inclusive extension with a join tree T such that S is precisely the set of all variables contained in the vertices of some subtree of T [1]. We call such a tree ext-S-connex tree. When S is the set of free variables of the CQ, and the CQ is clear from the context, we call such a tree *ext-free-connex*.

The notion of a *disruptive trio* has been introduced previously in the context of direct access to the answers of CQs [6]. A disruptive trio of a CQ $Q(\vec{x})$ is a set of three distinct free variables x_1, x_2, and x_3 such that x_1 and x_2 neighbor x_3 but not each other, and x_3 succeeds both x_1 and x_2 in $\vec{x}$. By saying that x and y are *neighbors* we mean that they occur jointly in at least one of the atoms.

Aggregate queries. By an *aggregate function* we refer to a function that takes as input a bag of tuples over Const and returns a single value in Const. We adopt the notation of Cohen

et al. [9] to our setting, as follows. An *aggregate query* here is an expression of the form

$$Q(\vec{x}, \alpha(\vec{w}), \vec{z}) :- \varphi_1(\vec{x}, \vec{y}, \vec{z}), \ldots, \varphi_\ell(\vec{x}, \vec{y}, \vec{z})$$

such that $Q'(\vec{x}, \vec{z}) :- \varphi_1(\vec{x}, \vec{y}, \vec{z}), \ldots, \varphi_\ell(\vec{x}, \vec{y}, \vec{z})$ is a CQ, α an aggregate function, and $\vec{w}$ a sequence of variables from $\vec{y}$. An example is $Q(x_1, x_2, \mathsf{Sum}(y_2), z) :- R(x_1, x_2, y_1), S(y_1, y_2, z)$. We refer to such a query as an *Aggregate CQ* or *ACQ* for short. Given a database D over Q', the result $Q(D)$ is defined by $Q(D) := \{(\vec{a}, \alpha(B(\vec{a}, \vec{b})), \vec{b}) \mid (\vec{a}, \vec{b}) \in Q'(D)\}$ where $B(\vec{a}, \vec{b})$ is the bag that is obtained by collecting the tuples $h(\vec{w})$ from every $h \in \mathsf{Hom}(Q', D)$ with $h(\vec{x}) = \vec{a}$ and $h(\vec{z}) = \vec{b}$. Note that our database and query model use set semantics, and we use bag semantics only to define the aggregate functions (in order to capture important functions such as count and sum).

We say that Q is *acyclic* if Q' is acyclic. Similarly, Q is *free-connex* if Q' is free-connex. A *disruptive trio* of Q is a disruptive trio of Q'; in other words, the definition of a disruptive trio remains unchanged when introducing aggregates, while we consider only the grouping variables and not the aggregate function.

▶ Remark 1. We remark on two aspects in our definition of ACQs. First, the reason for using both $\vec{x}$ and $\vec{z}$ as sequences of free variables is to determine a position for aggregate value $\alpha(\vec{w})$ and, consequently, define its position in the lexicographic order over the answers. Second, the reader should note that, in our notation, an ACQ has a single aggregate function. While this is important for some of our results, other results can be easily extended to multiple aggregate functions $\alpha(\vec{w_1}), \ldots, \alpha(\vec{w_k})$. We will mention this extension when relevant. ◀

In this work, we restrict the discussion to the common aggregate functions Count, CountD (count distinct), Sum, Avg (average), Min and Max. All aggregate functions take a single column as input (i.e., $\vec{y_i}$ is of length one) except for Count that counts the tuples in the group and takes no argument. For instance, the query Q_2 in the Introduction uses Count() and it could also use CountD(g) for counting the distinct games with scored goals.

Commutative semirings. A *commutative monoid* is an algebraic structure $(\mathbb{K}, \cdot)$ over a domain $\mathbb{K}$, with a binary operation $\cdot$ that satisfies *associativity*: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for any $a, b, c \in \mathbb{K}$, *commutativity*: $a \cdot b = b \cdot a$ for any $a, b \in \mathbb{K}$, and *identity element*: there exists an element $\varnothing \in \mathbb{K}$ such that $a \cdot \varnothing = a$ for any $a \in \mathbb{K}$. A *commutative semiring* is an algebraic structure $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ over a domain $\mathbb{K}$, with two binary operations $\oplus$ and $\otimes$ and two distinguished elements $\bar{0}$ and $\bar{1}$ in $\mathbb{K}$ that satisfy the following conditions: *(a)* $(\mathbb{K}, \oplus)$ is a commutative monoid with the identity element $\bar{0}$; *(b)* $(\mathbb{K}, \otimes)$ is a commutative monoid with the identity element $\bar{1}$; *(c)* $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ for all $a, b, c \in \mathbb{K}$; *and (d)* $a \otimes \bar{0} = \bar{0}$ for all $a \in \mathbb{K}$.

We refer to $\oplus$ as the *addition* operation, $\otimes$ as the *multiplication* operation, $\bar{0}$ as the *additive identity* and $\bar{1}$ as the *multiplicative identity*. We give examples of commutative semirings at the end of this section.

Annotated databases and query answers. Let $\mathbf{S}$ be a schema and $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ a commutative semiring. A $\mathbb{K}$-*database* (*over* $\mathbf{S}$) is a pair (D, τ) where D is a database over $\mathbf{S}$ and $\tau : D \to \mathbb{K}$ is function that maps every fact f of D to an element $\tau(f)$ of $\mathbb{K}$, called the *annotation* of f.

The annotation of a database propagates to the query answers by associating a semiring operation with each algebraic operation [13]. In the case of CQs, the relevant operations are *joins* and *projection*. For join, the annotation of the result is the product of the annotation

of the operands. For projection, the annotation is the sum of the annotations of the tuples that give rise to the result. In our terminology, we have the following.

Let $Q(\vec{x}) :- \varphi_1(\vec{x},\vec{y}),\ldots,\varphi_\ell(\vec{x},\vec{y})$ be a CQ and (D,τ) an annotated database. For a homomorphism h from Q to D, we denote by $\otimes h$ the product of the annotations of the facts in the range of h, that is $\otimes h := \tau(h(\varphi_1(\vec{x},\vec{y}))) \otimes \cdots \otimes \tau(h(\varphi_\ell(\vec{x},\vec{y})))$. An *answer* to Q over (D,τ) is a pair $(\vec{c},a)$ such that $\vec{c} \in Q(D)$ and

$$a = \oplus\{\otimes h \mid h \in \mathsf{Hom}(Q,D) \wedge h(\vec{x}) = \vec{c}\}$$

where, for $A = \{a_1,\ldots,a_n\} \subseteq \mathbb{K}$, we define $\oplus A = a_1 \oplus \cdots \oplus a_n$. As before, the *result* of Q over (D,τ), denoted $Q(D,\tau)$, is the set of answers $(\vec{c},a)$ to Q over (D,τ). We will make use of the fact that, over commutative semirings, projections and joins are commutative [13].

In this work, we study the ability to incorporate the annotation in the order over the answers. More precisely, we will investigate the complexity of involving the annotation in the lexicographic order over the answers, as if it were another value in the tuple. So, when we consider a CQ $Q(\vec{x})$, we need to specify where the annotation goes inside $\vec{x}$. Similarly to the ACQ notation, we do so by replacing $\vec{x}$ with a sequence $(\vec{x},\star,\vec{z})$ where $\star$ represents the annotation value. We refer to a CQ of this form as a CQ*. An example of a CQ* is $Q(x_1,x_2,\star,z) :- R(x_1,x_2,y_1), S(y_1,y_2,z)$ where the lexicographic order is by x_1, then by x_2, then by the annotation, and then by z.

Let Q be a CQ*, and let Q' be the CQ obtained from Q by removing $\star$ from the head. As in the case of ACQs, Q is *acyclic* if Q' is acyclic, Q is *free-connex* if Q' is free-connex, and a *disruptive trio* of Q is a disruptive trio of Q'.

Aggregate functions can often be captured by annotations of answers in annotated databases, where each aggregate function might require a different commutative semiring:

- Sum: the numeric semiring $(\mathbb{Q},+,\cdot,0,1)$.
- Count: the counting semiring $(\mathbb{N},+,\cdot,0,1)$.
- Max: the max tropical semiring $(\mathbb{Q}\cup\{-\infty\},\mathsf{max},+,-\infty,0)$.
- Min: the min tropical semiring $(\mathbb{Q}\cup\{\infty\},\mathsf{min},+,\infty,0)$.

The translation is straightforward (and known, e.g., [15, 20]), as we illustrate in Figure 1: the aggregated value becomes the annotation on one of the relations, the annotation outside of this relation is the multiplicative identity (as we later term "locally annotated"), and the addition operation captures the aggregate function. Note that in the case of the numeric and min/max tropical semirings, we are using the domain $\mathbb{Q}$ of rational numbers rather than all real numbers to avoid issues of numerical presentation in the computational model.

Avg can be computed using Sum and Count. CountD (count distinct) cannot be captured by a semiring, as the result of $\oplus$ cannot be computed from two intermediary annotations in the domain. We can, however, capture a semantically similar concept with the set semiring $(\mathcal{P}(\Omega),\cup,\cap,\varnothing,\Omega)$ by annotating each fact with the actual set of distinct elements. However, in such cases, we will need our complexity analysis to be aware of the cost of the operations.

3 The Direct-Access Problem

In this paper, we study CQs with lexicographic orders over the answers. As said earlier, the lexicographic order for the CQ $Q(\vec{x})$ is left to right according to $\vec{x}$. We will also investigate lexicographic orders that involve the annotation or aggregation when the query is a CQ* $Q(\vec{x},\star,\vec{z})$ or an ACQ $Q(\vec{x},\alpha(\vec{u}),\vec{z})$, respectively. We refer uniformly to the annotation of an answer (over an annotated database) and to the aggregate value of the answer's group (over an ordinary database) as the *computed value*.

Teams

p	c
1	5
2	5
3	6
4	7
5	8

Goals

g	p	t
1	1	31
1	3	50
1	3	75
2	4	90
2	4	9

Replays

g	t
1	1
1	31
1	50
2	5
1	90

$\Rightarrow$

Team

p	c	τ_+
1	5	1
2	5	1
3	6	1
4	7	1
5	8	1

Goals

g	p	t	τ_+
1	1	31	1
1	3	50	1
1	3	75	1
2	4	90	1
2	4	9	1

Replays

g	t	τ_+
1	1	1
1	31	31
1	50	50
2	5	5
1	90	90

Figure 1 An example of a $\mathbb{Q}$-database over the numerical semiring constructed to evaluate the ACQ $Q(c, \mathsf{Sum}(t)) :- \textsc{Teams}(p, c), \textsc{Goals}(g, p, t), \textsc{Replays}(g, t)$.

Let Q be a CQ, CQ*, or an ACQ. A *direct access* solution for Q consists of two algorithms: one for *preprocessing* and one for *access*.

- The preprocessing algorithm takes as input a database D over Q and constructs a data structure S_D.
- The access algorithm takes as input S_D and an index i, and returns the ith answer of $Q(D)$ in the lexicographic order. Note that this answer includes the computed value, when it exists. If $i > |Q(D)|$ then the algorithm should return *null*.

To define the complexity requirements of *efficient* direct access, we first describe the complexity model that we adopt. We use *data complexity* as a yardstick of tractability. Hence, complexity is measured in terms of the size of the database, while the size of the query is fixed (and every query is a separate computational problem). Assuming the input is of size n, we use the RAM model of computation with $O(\log n)$-bit words and uniform-cost operations. Notably, this model allows us to assume perfect hash tables can be constructed in linear time, and they provide access in constant time [12].

Let T_p and T_a be numeric functions. A direct-access algorithm is said to be *in* $\langle T_p, T_a\rangle$ if the preprocessing phase takes $O(T_p(|D|))$ time and each access takes $O(T_a(|D|))$ time. For example, $\langle \text{loglinear}, \log\rangle$ states that preprocessing constructs in $O(|D| \log |D|)$ time a data structure that provides $O(\log |D|)$-time access. In this work, a query Q has *efficient* direct access (and Q is deemed tractable) if it has a direct access algorithm in $\langle \text{loglinear}, \log\rangle$.

Carmeli et al. [6] established a dichotomy in the tractability of the CQs and lexicographic orders. This dichotomy relies on the following hypotheses.

- SparseBMM: two binary matrices $A^{n\times n}$ and $B^{n\times n}$ represented by lists of their non-zero entries cannot be multiplied in $O(m \operatorname{polylog}(m))$ time where m is the total number of non-zero entries in A, B and $A \times B$.
- HYPERCLIQUE: for all $k \geq 2$, there is no $O(m \operatorname{polylog}(m))$-time algorithm that, given a hypergraph with m hyperedges, determines whether there exists a set of $k+1$ vertices such that every subset of k vertices among them forms a hyperedge.

▶ **Theorem 2** ([6]). *Let Q be a CQ.*

1. *If Q is free-connex with no disruptive trio, then direct access for Q is in $\langle \text{loglinear}, \log\rangle$.*
2. *Otherwise, if Q is also self-join-free, then direct access for Q is not in $\langle \text{loglinear}, \log\rangle$, assuming the HYPERCLIQUE hypothesis (in case Q is cyclic) and the SparseBMM hypothesis (in case Q is acyclic).*

4 Incorporating Annotation and Aggregation in the Answers

In this section, we discuss the existence of efficient direct access in the case where the order *does not* involve the computed value, that is, the annotation (for CQ*s) or the aggregate values (for ACQs). Equivalently, these are queries where the computed value is last in order, that is, the vector $\vec{z}$ in the head is empty. Hence, we focus on CQ*s of the form $Q(\vec{x}, \star)$ and ACQs of the form $Q(\vec{x}, \alpha(\vec{w}))$. In other words, the problem is similar to the CQ case, except that the access algorithm should also retrieve the aggregated value from the data structure. In Section 4.1, we will use annotated databases to identify the cases where this can be done efficiently for min, max, count, sum, and average. By contrast, in Section 4.2 we will show that we cannot do the same for count-distinct, even in the case of an extremely simple query, unless the domain of the elements we count is small (logarithmic-size).

But first, we need to be clear about the complexity of the semiring operations. The RAM model allows us to assume that the numeric, counting, min tropical, and max tropical semirings use constant space for representing values and constant time for the operations $\oplus$ and $\otimes$. In fact, it suffices for our results to assume that the operations take logarithmic time, and later we will make use of this relaxed assumption (within a special case of CountD). We refer to a semiring with this property as a *logarithmic-time (commutative) semiring*.

In our proofs, we will use the following definition of when two facts f and f' agree in the context of two queries. Intuitively, facts agree if they assign the same variables with the same values.

▶ **Definition 3.** *Let Q and Q' be CQs over the schemas $\mathbf{S}$ and $\mathbf{S}'$, respectively. Let φ and φ' be atoms of Q and Q' over the relation symbols R and R', respectively. Let f and f' be facts over R and R', respectively. We say that f and f'* agree *(w.r.t. φ and φ') if there exists a homomorphism $h : \mathrm{vars}(\phi) \cup \mathrm{vars}(\phi') \to \mathsf{Const}$ such that f and f' are obtained from φ and φ', respectively, by replacing each variable x with the constant $h(x)$.*

4.1 Generalized Dichotomies

We now show that Theorem 2 extends to databases with annotations, and so, also to queries with aggregate functions that can be efficiently simulated by annotations. The first step is to eliminate the existential variables from the CQ*. It is folklore that free-connex CQs (over non-annotated databases) can be transformed into *full* acyclic CQs in linear time [14,18]. The following lemma states that the same holds for free-connex CQ*s over annotated databases.

▶ **Lemma 4.** *Let $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ be a logarithmic-time commutative semiring, and let $Q(\vec{x}, \star, \vec{z})$ be a free-connex CQ*. There exists a full acyclic CQ* $Q'(\vec{x}, \star, \vec{z})$, without self-joins, and an $O(|D| \log |D|)$-time algorithm that maps $\mathbb{K}$-databases (D, τ) of Q to $\mathbb{K}$-databases (D', τ') of Q' such that* (a) *$Q'(D', \tau') = Q(D, \tau)$;* (b) *the variables in every atom of Q' are contained in an atom of Q;* and (c) *Q has a disruptive trio if and only if Q' has a disruptive trio.*

We note that in the case where the semiring operations require only constant time, the algorithm of Lemma 4 runs in linear time instead of loglinear time. With this lemma, we can now prove the following generalization of Theorem 2 to annotated databases.

▶ **Theorem 5.** *Let $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ be a logarithmic-time commutative semiring, and let $Q(\vec{x}, \star)$ be a CQ*.*

1. *If Q is free-connex and with no disruptive trio, then direct access for Q is in $\langle \mathrm{loglinear}, \log \rangle$ on $\mathbb{K}$-databases.*

2. *Otherwise, if Q is also self-join-free, then direct access for Q is not in $\langle \text{loglinear}, \log \rangle$, assuming the HYPERCLIQUE hypothesis (in case Q is cyclic) and the SparseBMM hypothesis (in case Q is acyclic).*

Proof. For the negative side of the dichotomy, we simply use the negative side of Theorem 2. This can be done since each answer to a CQ* contains the ordinary (non-annotated) answer to the CQ obtained by removing $\star$, and the answers have the same order. It is left to prove the positive side of the dichotomy.

We can use Lemma 4 to focus on full CQ*s without self-joins. We build on the algorithm that Carmeli et al. [6] presented for proving what we gave here as Theorem 2. This algorithm uses the concept of a *layered join tree*, defined for a CQ Q. It can be seen as a particular kind of join tree of a query $Q_{\texttt{lay}}$ equivalent to Q. That is, the variables of every atom of Q are contained in an atom of $Q_{\texttt{lay}}$ and vice versa. They showed how to find such a tree when there is no disruptive trio. They also showed how, given a database D over Q, to construct a database $D_{\texttt{lay}}$ over $Q_{\texttt{lay}}$ such that $Q(D) = Q_{\texttt{lay}}(D_{\texttt{lay}})$ in $O(|D| \log |D|)$ time. Then, they showed how to use the special structure of the layered join tree to perform access calls in logarithmic time. In an access call, a fact is selected from each relation of $D_{\texttt{lay}}$, and these facts are joined to form the answer.

We use the same construction and incorporate the annotations as follows. We construct $Q_{\texttt{lay}}$ and $(D_{\texttt{lay}}, \tau_{\texttt{lay}})$ from Q and (D, τ), respectively. For $Q_{\texttt{lay}}$ and $D_{\texttt{lay}}$, we use the same construction as that of Carmeli et al. [6], and we apply it by ignoring the annotation. Next, we annotate each fact f of $D_{\texttt{lay}}$ with the initial value $\tau'(f) = \bar{1}$, and then apply the following operation.

1: **for all** atoms φ of Q **do**
2: Select an atom $\varphi_{\texttt{lay}}$ of $Q_{\texttt{lay}}$ such that $\text{vars}(\varphi) \subseteq \text{vars}(\varphi_{\texttt{lay}})$
3: Let R and $R_{\texttt{lay}}$ be the relation symbols of φ and $\varphi_{\texttt{lay}}$, respectively
4: **for all** facts $f_{\texttt{lay}}$ of $R_{\texttt{lay}}$ **do**
5: Find a fact f of R such that f and $f_{\texttt{lay}}$ agree w.r.t. φ and $\varphi_{\texttt{lay}}$ (see Definition 3)
6: $\tau_{\texttt{lay}}(f_{\texttt{lay}}) \leftarrow \tau_{\texttt{lay}}(f_{\texttt{lay}}) \cdot \tau(f)$

Note that in line 5, at most one corresponding f exists for every $f_{\texttt{lay}}$ since $\text{vars}(\varphi) \subseteq \text{vars}(\varphi_{\texttt{lay}})$. Moreover, from the construction of Carmeli et al. [6] it follows that such an f necessarily exists (since they apply the full reduction of the Yannakakis [23] algorithm). Finding each f can be done in constant time by constructing a hash table where each key is a tuple of values assigned to $\text{vars}(\varphi)$ by f; then, for each $f_{\texttt{lay}}$, we project out other variables and search the hash table. In total, this procedure can be done in loglinear time.

The access algorithm extends naturally from before: a fact is selected from each relation of $D_{\texttt{lay}}$, and those are combined to form an answer. The annotation of the answer is the product of the annotations of the selected facts. Retrieving the answer takes logarithmic time, and then we compute the annotation in logarithmic time using a constant number of semiring multiplications. The returned answer is annotated correctly: the annotation is indeed the product of the annotations of the facts of D that form the answer, and those were multiplied to form the annotation in $D_{\texttt{lay}}$. ◀

From the positive side of Theorem 5, we conclude efficient direct access for ACQs Q, as long as we can efficiently formulate the aggregate function as an annotation over some logarithmic-time commutative semiring. This is stated in the following corollary of Theorem 5.

▶ **Corollary 6.** *Consider an ACQ $Q(\vec{x}, \alpha(\vec{w})) :- \varphi_1(\vec{x}, \vec{y}), \ldots, \varphi_\ell(\vec{x}, \vec{y})$ where α is one of* Min, Max, Count, Sum, *and* Avg.

1. *If the CQ* $Q'(\vec{x}) :- \varphi_1(\vec{x},\vec{y}),\ldots,\varphi_\ell(\vec{x},\vec{y})$ *is free-connex with no disruptive trio, then direct access for* Q *is in* $\langle \text{loglinear}, \log\rangle$.
2. *Otherwise, if* Q *is also self-join-free, then direct access for* Q *is not in* $\langle \text{loglinear}, \log\rangle$*, assuming the* HYPERCLIQUE *hypothesis (in case* Q *is cyclic) and the* SparseBMM *hypothesis (in case* Q *is acyclic).*

Proof. For the positive side, we simply apply Theorem 5 with the corresponding semiring. In the case where α is Avg, we compute Sum and Count separately and divide the results. The negative side carries over from Theorem 2 since a direct access solution for Q in $\langle \text{loglinear}, \log\rangle$ is also a direct access solution for Q' in $\langle \text{loglinear}, \log\rangle$ if we ignore the aggregated values. ◀

▶ Remark 7. Corollary 6 can be easily extended to support multiple aggregate functions $\alpha_1(\vec{w}_1)$, …, $\alpha_k(\vec{w}_k)$. For that, we can simply solve the problem for each $\alpha_i(\vec{w}_i)$ separately, and extract the aggregate values of an answer from the k data structures that we construct in the preprocessing phase. (Moreover, a practical implementation can handle all aggregate values in the same structure.) ◀

4.2 Hardness of Count Distinct

Can we generalize Corollary 6 beyond the stated aggregate functions? The most notable missing aggregate function is CountD (count distinct). Next, we show that we *cannot* have similar tractability for count distinct, even in the case of a very simple query, under the *small-universe Hitting Set Conjecture (HSC)* [22]. In HSC, we are given two sets $\mathcal{U}$ and $\mathcal{V}$ of size N, each containing sets over the universe $\{1, 2, \ldots, d\}$, and the goal is to determine whether $\mathcal{U}$ contains a set that shares an element with (hits) every set in $\mathcal{V}$. HSC states that the problem takes $N^{2-o(1)}$ time for every function $d = \omega(log(N))$. (In fact, it is conjectured that even a randomized algorithm for this problem needs $N^{2-o(1)}$ time in expectation [22].)

▶ **Theorem 8.** *Direct access for* $Q(x, \mathsf{CountD}(y)) :- R(x,w), S(y,w)$ *is not in* $\langle \text{loglinear}, \log\rangle$*, assuming HSC.*

Proof. Let $\mathcal{U} = \{U_1, U_2, \ldots, U_N\}$ and $\mathcal{V} = \{V_1, V_2, \ldots, V_N\}$ be sets of sets of elements of the universe $\{1, 2, \ldots, d\}$ where $d = N^c$ for some $0 < c < 1$. Indeed, $d = \omega(logN)$, as required by the conjecture. We construct the database D over R and S with the fact $R(i, j)$ for all $j \in U_i$ and $S(i, j)$ for all $j \in V_i$.

Next, we assume efficient direct access for $Q(x, \mathsf{CountD}(y)) :- R(x,z), S(y,z)$, and use that to solve the hitting-set problem. Each query answer is a pair (i, c) where i is the index of a set U_i from $\mathcal{U}$ and c is the number of sets V_j that U_i hits. By accessing all query answers, we can check all sets in $\mathcal{U}$, one by one, and see whether any is hitting all N sets. The number of facts in D is $O(dN)$, and the number of query answers (and so the number of access calls) is N. Hence, direct access for Q in $\langle \text{loglinear}, \log\rangle$ would imply a solution to the hitting-set problem in better than $N^{2-o(1)}$ time. Indeed, for any $d = O(N^c)$, we get a solution in $O(N^{1+c} \cdot \log N)$ time, which contradicts HSC for $c < 1$. ◀

Importantly, the reduction used in the proof of Theorem 8 does not involve the order, and hence, the theorem holds true even for direct access without any order requirement.

Despite the above example of intractability, it is important to observe that there are cases where Theorem 5 can be used to compute count distinct: when the size of the domain Ω of the distinct elements we count is bounded by a logarithm in the input size $|D|$. Such an assumption can be realistic when we count, say, distinct categories from a small ontology (e.g., item categories in a sales context), distinct countries from a small collection of countries,

and so on. In such cases, we can compute the exact set of distinct elements, and not just their count, by using the set semiring $(\mathcal{P}(\Omega), \cup, \cap, \varnothing, \Omega)$ since the operations $\cup$ and $\cap$ can be performed in logarithmic time for logarithmic domains.

5 Incorporating the Annotation and Aggregation in the Order

The results of the previous section apply when the lexicographic order does not include the computed value, or equivalently when the computed value is last in the head of the query. In this section, we explore the ability to include the computed value earlier in the lexicographic order. To this end, we assume that the underlying commutative semiring has an ordered domain. When the domain is numerical, we will implicitly assume the natural order without mentioning it. In terms of the computational model, we assume that we can compare two given elements of the domain in time logarithmic in the input.

We first discuss the hardness encountered when we desire to incorporate the computed value in the order. It turns out that this hardness is hit already in extremely simple queries. This is a contrast to the case of Section 4 when this value is excluded from the order.

Hardness of a CQ*. Consider the simplest possible Cartesian-product query: $R \times S$ for unary R and S. We wish to have the annotation *first* in the order, hence we have the CQ*

$$Q_{\star\times}(\star, x, y) :- R(x), S(y)\,. \tag{1}$$

The next theorem states that under the *3SUM conjecture*, direct access for $Q_{\star\times}$ is impossible over $\mathbb{K}$-database with semirings that gave positive results in the previous section. The 3SUM conjecture [11, 19] states it takes $N^{2-o(1)}$ time to determine whether a given set of N elements from $\{-N^3, \ldots, N^3\}$ contains distinct elements a, b, c such that $a + b = c$.

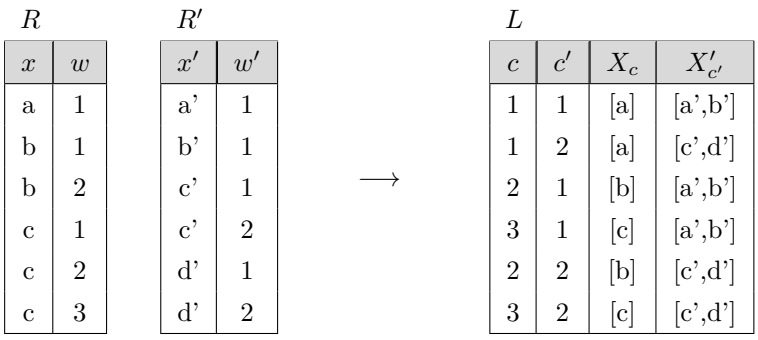
▶ **Theorem 9.** *Let $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ be one of the counting, numerical, max tropical, or min tropical semirings. Direct access for the CQ* $Q_{\star\times}$ (of Equation (1)) is not in $\langle \text{loglinear}, \log\rangle$ over $\mathbb{K}$-databases, assuming the 3SUM conjecture.*

The proof of Theorem 9 shows how to solve 3SUM using an algorithm for $Q_{\star\times}$. The proof is nontrivial and is more involved in the case of the counting and numerical semirings, where we use results from number theory [8] to define a homomorphism, such that the operation $\otimes$ of the semiring represents the numerical addition of 3SUM.

Hardness of an ACQ. Theorem 9 states the hardness of direct access for $R \times S$ by the order of the annotation. For that, we needed to use the power of the annotation, namely, the annotation of an answer (a, b) is the product of the annotations of $R(a)$ and $S(b)$. This does not necessarily imply that we have a similar hardness when the computed value is within an ACQ, say using Count. For example, direct access for $Q(\mathsf{Count}(), x, y) :- R(x), S(y)$ is clearly in $\langle \text{loglinear}, \log\rangle$ since the computed value has no impact (as it is always 1).

Nevertheless, we can show that incorporating the computed value in the order introduces hardness for another fixed ACQ $Q_c(\mathsf{Count}(), x, y)$. Moreover, this ACQ is tractable if the order was $x, y, \mathsf{Count}()$, due to Theorem 5. We prove it using a reduction from $Q_{\star\times}$ with the counting semiring $(\mathbb{N}, +, \cdot, 0, 1)$.

▶ **Theorem 10.** *There exists a free-connex ACQ $Q_c(\mathsf{Count}(), x, y)$ such that direct access for Q is not in $\langle \text{loglinear}, \log\rangle$, assuming the 3SUM conjecture.*



$$Q(\mathsf{Count}(), x, x') :- R(x, w), R'(x', w')$$

Figure 2 Example of the construction in the proof of Proposition 11: direct access for the ACQ $Q(\mathsf{Count}(), x, x') :- R(x, w), R'(x', w')$.

In Theorem 9, we stated hardness for the specific CQ* $Q_{\star\times}(\star, x, y) :- R(x), S(y)$, while in Theorem 10 we only claimed the existence of the hard ACQ $Q_c(\mathsf{Count}(), x, y)$ since the latter is more involved (and we construct it in the proof). The reader might wonder whether we could also phrase Theorem 10 over the Cartesian product $Q(\mathsf{Count}(), x, y) :- R(x), S(y)$ or alike. Clearly, incorporating Count in $R(x) \times S(y)$ would be meaningless since every answer appears exactly once (and has a count of 1). The next theorem shows that the reason goes deeper: even if we add to $Q(\mathsf{Count}(), x, y) :- R(x), S(y)$ existential variables, the query remains in $\langle \text{loglinear}, \log \rangle$. This statement, in contrast to $Q(\mathsf{Count}(), x, y) :- R(x), S(y)$, is nontrivial and requires a proof.

▶ **Proposition 11.** *For $Q(\mathsf{Count}(), x, y) :- R(x, w), S(y, z)$, direct access is in $\langle \text{loglinear}, \log \rangle$.*

Proof. For ease of notation, we rename Q as follows:

$$Q(\mathsf{Count}(), x, x') :- R(x, w), R'(x', w')$$

In the remainder of this proof, we fix an input database D for Q. Note that the answers are of the form $(c \cdot c', a, a')$ where c is the count of the facts $R(a, \cdot)$ that have a as the first element, and c' is the count of the facts $R'(a', \cdot)$ that have a' as the first element.

Let us describe the preprocessing step. First, we compute the number of facts $R(a, \cdot)$ for every possible value of a. We use the result to create the set of all counts per possible value of x, and denote this set by C. For all $c \in C$, we also keep a list X_c with the set of all values a that appear c times in the left column of R. The list X_c is sorted so that we can easily locate a given a. We do the same for R' to obtain C' and a sorted list $X'_{c'}$ for every $c' \in C'$.

Next, we create a list L, where for every pair of counts $(c, c') \in C \times C'$ it holds the tuple $(c, c', X_c, X'_{c'})$ where X_c and $X'_{c'}$ are represented as *pointers* to the corresponding lists. See Figure 2 for an example of the construction of L from an input database. We perform direct access on L sorted by $c \cdot c'$ and weighted by $|X_c| \cdot |X'_{c'}|$ using prefix sum, similarly to the way established by Carmeli et al. [6] for a single relation, as we briefly describe next.

We first sort L by the product of the first two elements of each tuple, $c \cdot c'$. Notice that L indeed represents the answers in the order we desire. For example, if the first fact in L is $(c, c, X_c, X'_{c'})$, then the first $|X_c| \cdot |X'_{c'}|$ answers have the count $c \cdot c'$ and assign to x and x' the values that occur in X_c and $X'_{c'}$, respectively. We then iterate over L, and for a tuple index i we compute the sum of $|X_c| \cdot |X'_{c'}|$ over all tuples in indices $1, \ldots, i-1$ in L. We denote this sum by l_i. Note that $l_{i+1} > l_i$ for all $i = 1, \ldots, |L|$.

We now describe the access procedure. Suppose that we are requested to fetch result number d. We perform a binary search on L to find the tuple in index i such that $l_i < d \leq l_{i+1}$. Assume that this tuple is $(c, c', X_c, X'_{c'})$. The count we return is $c \cdot c'$. Next, we need to access the $(d - l_i)$th element (x, x') in $X_c \times X'_{c'}$, sorted lexicographically by x and then by x'. As in multidimensional arrays, we assign to x' the element with the index $(d - l_i) \mod |X'_{c'}|$ of $X'_{c'}$, and we assign to x the element with the index $\lfloor \frac{d-l_i}{|X'_{c'}|} \rfloor$ of X_c.

We now analyze the execution time. Processing each of the a counts and the a' counts separately requires only $O(|D|)$ time. The concern is the time it takes to build and sort the list L, which might be of size $|C| \cdot |C'|$. Assume, without loss of generality, that $|C| \geq |C'|$. We claim that $|R^D| = \Omega(|C|^2)$. If R^D has the smallest possible number of facts to result in $|C|$ distinct counts, then the counts are $1, \ldots, |C|$. The number of facts in this case is $\sum_{i=1}^{|C|} i = \frac{|C|(|C|+1)}{2}$. So, $|L| = |C| \cdot |C'| \leq |C|^2 = O(|R^D|)$. We conclude that the algorithm runs in $O(|D| \log |D|)$ preprocessing time and $O(\log |D|)$ access time. In conclusion, direct access for Q is in $\langle \text{loglinear}, \log \rangle$, as claimed. ◀

5.1 Tractability Condition for General Semirings

So far, we have seen examples where it is intractable to incorporate the computed value in the order. *Not* incorporating it is the same as positioning it last in the lexicographic order. In this section, we show that we can be flexible about the position, to some extent, and pull the computed value back to an earlier position. For example, we show that direct access for the following CQ* is in $\langle \text{loglinear}, \log \rangle$ over databases annotated with the numerical semiring.

$$Q(w, x, \star, y, z) :- R(w, x), S(x, y, z), T(y, z) \tag{2}$$

Let $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ be a commutative semiring in a domain $\mathbb{K}$ with an underlying order $\succeq$. The semiring is said to be $\otimes$-*monotone* if the function f_c is monotone for every $c \in \mathbb{K}$, where $f_c : \mathbb{K} \to \mathbb{K}$ is defined by $f_c(y) = c \otimes y$. This means that either $c \otimes a \succeq c \otimes b$ whenever $a \succeq b$, or $c \otimes a \succeq c \otimes b$ whenever $b \succeq a$. All specific semirings that we mention in the paper are $\otimes$-monotone. Computationally, we assume that we can determine efficiently (in logarithmic time in the input) whether a given c is such that the function $f_c(x) = c \otimes x$ is non-decreasing or non-increasing.

▶ **Theorem 12.** *Let $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ be a $\otimes$-monotone logarithmic-time commutative semiring, and $Q(\vec{x}, \star, \vec{z})$ a free-connex CQ* with no disruptive trio. If every atom of Q contains either all variables of $\vec{z}$ or none of them, then direct access for Q is in $\langle \text{loglinear}, \log \rangle$.*

As an example, consider again the CQ* of Equation (2). The variables that follow the computed value $\star$ are y and z, and indeed, every atom either contains both y and z (as the second and third atoms) or contains none of them (as the first atom). Hence, direct access is tractable according to Theorem 12. As another example, recall the intractable CQ* $Q_{\star\times}(\star, x, y) :- R(x), S(y)$ from Theorem 9. Note that this is *not* one of the tractable cases of Theorem 12 since there is an atom that contains x but not y. In contrast, Theorem 12 does indicate that whenever $\star$ is not first, as is the case with $(x, \star, y)$, the query is tractable.

Similarly to Corollary 6, we conclude the following corollary for ACQs.

▶ **Corollary 13.** *Let $Q(\vec{x}, \alpha(\vec{w}), \vec{z}) :- \varphi_1(\vec{x}, \vec{y}), \ldots, \varphi_\ell(\vec{x}, \vec{y})$ be a free-connex ACQ with no disruptive trio. Suppose that α is one of* Min*,* Max*,* Count*,* Sum*, and* Avg*. If every atom in Q contains either all or none of the variables of $\vec{z}$, then direct access for Q is in $\langle \text{loglinear}, \log \rangle$.*

In the next section, we study how additional assumptions on the annotated database can lead to additional opportunities to efficiently incorporate the computed value in the ordering.

5.2 Locally Annotated Databases

We have seen in Theorem 9 that even the simple CQ* $Q_{\star\times}(\star, x, y) :- R(x), S(y)$ is intractable. This hardness does not necessarily apply to ACQs. For illustration, consider the ACQ

$$Q(\mathsf{Sum}(w), x, y) :- R(x, w), S(y)\,. \tag{3}$$

When translating into an annotated database, we obtain the CQ* $Q_{\star\times}$ over $\mathbb{Q}$-databases annotated by the numerical semiring $(\mathbb{Q}, +, \cdot, 0, 1)$. Hence, we translate the problem into an intractable one. Nevertheless, direct access for Q by $(\star, x, y)$ is, in fact, in $\langle \text{loglinear}, \log \rangle$, as we will show in Theorem 17. This discrepancy stems from the fact that the hardness of $Q_{\star\times}$ (established in the proof of Theorem 9) relies on the annotation of tuples from both R and S. Yet, in our translation, all S-facts are annotated by 1, and only R-facts have a nontrivial annotation. The resulting $\mathbb{K}$-database is such that every fact is annotated by $\bar{1}$ (the multiplicative identity), with the exception of one relation. We call such a $\mathbb{K}$-database *locally annotated* or *R-annotated* (when we need to specify R). We now focus on such databases and show how the assumption of local annotations can be used for efficient access.

In the remainder of this section, we restrict the discussion to queries without self-joins[1] and fix a logarithmic-time commutative semiring $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$.

Full CQ*s. We first discuss full CQ*s (i.e., without existential variables). In the next sections, we also discuss the implications on (non-full) ACQs. We begin with some examples that demonstrate the results that follow later in this section.

▶ **Example 14.** In some cases, incorporating the annotation in an otherwise tractable order may introduce hardness. Let $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ be a logarithmic-time commutative semiring, and consider the full CQ*

$$Q(\vec{x}, \star, \vec{z}) :- R(x_1, x_3), S(x_2, x_3)$$

over S-annotated $\mathbb{K}$-databases. Note that Q has a disruptive trio if x_3 appears after both x_1 and x_2 in the order. Let us consider orders where this is not the case. The lexicographic order $(\star, x_2, x_3, x_1)$ is not covered by Theorem 12, as x_1 appears in R, but x_2 does not. However, we will show in Theorem 17 that, when considering S-annotated $\mathbb{K}$-databases, direct access for Q by this order is in $\langle \text{loglinear}, \log \rangle$, while direct access for Q by $(\star, x_1, x_3, x_2)$ is *not* in $\langle \text{loglinear}, \log \rangle$. ◀

▶ **Example 15.** It may also happen that we incorporate the annotation non-trivially and the query remains tractable. Consider the full CQ*

$$Q'(\vec{x}, \star, \vec{z}) :- R(x_1, x_3), S(x_2, x_3), T(x_3)$$

[1] In fact, for the algorithm, it suffices that the relation with unrestricted annotations will not appear in more than one atom.

over T-annotated $\mathbb{K}$-databases. Note that this case is a slight variation on Example 14. For any lexicographic order $(\vec{x}, \star, \vec{z})$, if x_3 appears after x_1 and x_2, then Q' has a disruptive trio and, therefore, direct access for Q' is not in $\langle \text{loglinear}, \log \rangle$. As we show in Theorem 17, for Q' over T-annotated $\mathbb{K}$-databases, that lack of a disruptive trio is a sufficient condition for tractability. That is, if x_3 does not appear after x_1 and x_2, then for any aggregate function α it holds that direct access for Q' (and for Q) is in $\langle \text{loglinear}, \log \rangle$. ◀

When dealing with R-annotated databases, we can replace the annotation of the facts of R with a new extra attribute, added to R, and then reason about orders that involve the annotation by considering orders that involve the new attribute instead. To do so, we introduce the following variations of a query and order. Let Q be a full acyclic CQ* without self-joins. For a relation symbol R of Q, we define the *R-deannotation* of $Q(\vec{x}, \star, \vec{z})$ to be the CQ Q_R obtained as follows, where we denote by ϕ_S the atom of a relation symbol S.

- In the head, replace $\star$ with a new variable y.
- If, in addition, $\text{vars}(\phi_R)$ contains only variables from $\vec{x}$, then in the head of Q_R, advance y to be immediately after the last variable of ϕ_R.
- For each relation S of Q, if $\text{vars}(\phi_R) \subseteq \text{vars}(\phi_S)$ then concatenate y to the variable sequence of ϕ_R.

▶ **Example 16.** Consider the CQ* Q' of Example 15. The R-deannotation of Q' is

$$Q_R(\vec{x}_R, y, \vec{z}_R) :- U(x_1, x_3, y), V(x_2, x_3, y), R(x_3, y)$$

where $\vec{x}_R$ and $\vec{z}_R$ are adjustments of $\vec{x}$ and $\vec{z}$: if x_3 is in $\vec{x}$, the suffix of $\vec{x}$ that follows x_3 is moved to the beginning of $\vec{z}$. ◀

When Q is full, we can reduce direct access for Q to direct access for Q_R. This is stated in the next theorem. The theorem also says that, whenever the annotation domain contains the natural numbers, this reduction is optimal in the sense that, if we got an intractable Q_R, then direct access for Q was hard to begin with.

▶ **Theorem 17.** *Let $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ be a logarithmic-time commutative semiring. Let Q be a full CQ* without self-joins and Q_R the R-deannotation of Q for a relation symbol R of Q.*

1. *If Q_R is acyclic and has no disruptive trio, then direct access for Q is in $\langle \text{loglinear}, \log \rangle$ on R-annotated $\mathbb{K}$-databases.*
2. *Otherwise, if $\mathbb{N} \subseteq \mathbb{K}$, then direct access for Q is not in $\langle \text{loglinear}, \log \rangle$ on R-annotated $\mathbb{K}$-databases, assuming the HYPERCLIQUE hypothesis (in case Q_R is cyclic) and the SparseBMM hypothesis (in case Q_R is acyclic).*

We obtain the positive side of the result by treating the annotation as an extra variable that depends functionally on the original variables of R. Our definition of the R-deannotation is exactly the *FD-reordered extension* [6] of the query with this extra variable, and the tractability of such an extension is known to imply the tractability of the original query. We note that the negative side of Theorem 17 applies to any domain other than $\mathbb{N}$, as long as we can generate infinitely many elements according to the underlying order of the semiring.

▶ **Example 18.** Theorem 17 gives us a useful tool to analyze the previous examples. Recall that in Example 16 we showed the R-deannotation of Q' from Example 15. Since y appears in every atom, it cannot be part of any disruptive trio. In particular, the disruptive trios of Q_R are exactly the disruptive trios of Q. Therefore, given an order $(\vec{x}, \alpha(w), \vec{z})$, checking whether direct access for Q is in $\langle \text{loglinear}, \log \rangle$ boils down to verifying that x_1, x_2 and x_3 do not form a disruptive trio. ◀

General Queries in the Case of Idempotence. Next, we extend Theorem 17 beyond full CQ⋆s. In some cases, it is sufficient to assemble the tools we already established. Consider the following ACQ:

$$Q(\mathsf{Max}(w_2), x_1, x_2, x_3) :- R(x_1, x_3, w_3), S(x_2, x_3), T(x_3, w_1), U(w_1, w_2) \tag{4}$$

We can solve direct access for Q using direct access for the CQ⋆

$$Q'(\star, x_1, x_2, x_3) :- R(x_1, x_3, w_3), S(x_2, x_3), T(x_3, w_1), U'(w_1)$$

over U'-annotated $\mathbb{Q}$-databases and the max tropical semiring. We can then use Lemma 4 to eliminate existential variables and reduce direct access for Q' to direct access for a full CQ⋆ Q_{full}. As we later prove in Lemma 19, when the input database for Q' is U'-annotated over the max tropical semiring, the suitable database for Q_{full} is T'-annotated for a relation symbol T' of Q_{full}. In our case, we obtain

$$Q_{\mathsf{full}}(\star, x_1, x_2, x_3) :- R(x_1, x_3), S(x_2, x_3), T'(x_3)$$

and a T'-annotated database. From Q_{full} we define Q_0 as the T'-deannotation of Q_{full}.

$$Q_0(y, x_1, x_2, x_3) :- R(x_1, x_3, y), S(x_2, x_3, y), T'(x_3, y)$$

Theorem 2 determines that direct access for Q_0 is in $\langle \text{loglinear}, \log \rangle$, and so we can use Theorem 17 to deduce that Q_{full} is in $\langle \text{loglinear}, \log \rangle$ on T'-annotated $\mathbb{Q}$-databases. Therefore, we know from Theorem 5 that direct access for Q' on U-annotated $\mathbb{Q}$-databases is in $\langle \text{loglinear}, \log \rangle$ and as a consequence so is direct access for Q.

As we explain next, the argument above is specific to Max and not all aggregate functions since we rely on Max being *idempotent*. An operation $\oplus$ is said to be idempotent if for every a in the domain $\mathbb{K}$ we have that $a \oplus a = a$. We say that a commutative semiring is an $\oplus$-*idempotent semiring* if its addition operation, $\oplus$, is idempotent.

Let $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ be some logarithmic-time commutative semiring. The process of existential variable elimination using Lemma 4 takes a free-connex CQ⋆ Q and a $\mathbb{K}$-database (D, τ) and translates it to a full acyclic CQ⋆ Q' and a $\mathbb{K}$-database (D', τ'). When working over an $\oplus$-idempotent semiring, if the input database is locally annotated, then the output database is also guaranteed to be locally annotated. The semirings used for CountD, Min, and Max are $\oplus$-idempotent and therefore provide such a guarantee, while the semirings used for Sum and Count do not. In the case of the query of Equation (4), if the aggregate function was Sum instead of Max, once we eliminate existential variables, the database for Q_{full} is no longer guaranteed to be locally-annotated since projecting out the existential variable w_3 may cause R, in addition to T', to be annotated with values other than $\bar{1}$.

Using Lemma 4 to eliminate existential variables does not guarantee optimal results, even over $\oplus$-idempotent semirings. Consider the following simplified version of Equation (4):

$$Q(\mathsf{Max}(w_2), x_1, x_2, x_3) :- U(x_1, x_3), V(x_2, x_3), R(x_3, w_1, w_2) \tag{5}$$

Lemma 4 eliminates existential variables using any ext-free-connex tree and outputs a new full CQ⋆ Q_{full}. In this process, the vertex of R is eliminated, and the annotations its relation contained are reflected in a different relation. Then, if Theorem 17 indicates that direct access for Q_{full} is in $\langle \text{loglinear}, \log \rangle$, we can use Q_{full} to provide direct access to Q. Figure 3 describes three ext-free-connex trees for Q. The choice between them for usage in Lemma 4 is important:

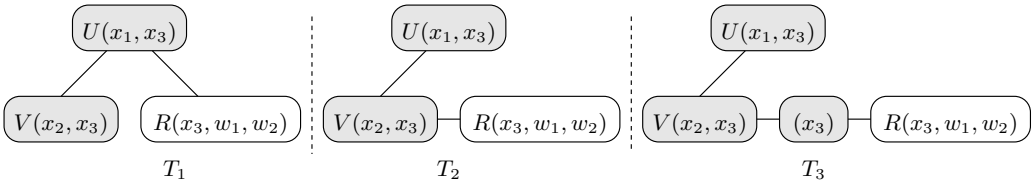


Figure 3 Possible ext-free-connex trees of $Q(x_1, x_2, x_3) :- U(x_1, x_3), V(x_2, x_3), R(x_3, w_1, w_2)$. The subtree that contains exactly the free variables appears in gray.

- If we use the tree T_1, the annotations would be reflected in the relation of U. Using Theorem 17 we get that direct access for Q_{full} on U-annotated $\mathbb{K}$-databases is not in $\langle \text{loglinear}, \log \rangle$. So, this tree cannot be used to obtain direct access for Q in $\langle \text{loglinear}, \log \rangle$.
- If we use the tree T_2, the annotations would be reflected in the relation of V and we get that direct access for Q_{full} on V-annotated $\mathbb{K}$-databases is not in $\langle \text{loglinear}, \log \rangle$. So, this tree cannot be used to obtain direct access for Q in $\langle \text{loglinear}, \log \rangle$ either.
- Only by choosing tree T_3 would Lemma 4 admit that direct access to Q_{full} is in $\langle \text{loglinear}, \log \rangle$. In this case, the new relation that corresponds to (x_3) would reflect the annotations, and Theorem 17 would indicate that direct access is in $\langle \text{loglinear}, \log \rangle$.

With these examples in mind, we establish an effective classification in the case of locally annotated databases over an idempotent semiring. This will be given in Theorem 20. To prove it, we use the next lemma that enables transforming a CQ* with existential variables into a full CQ*. Note that this lemma is different from Lemma 4 in the sense that the translation preserves intractability in addition to tractability (assuming that the semiring is $\oplus$-idempotent).

▶ **Lemma 19.** *Let $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ be a logarithmic-time $\oplus$-idempotent commutative semiring. There exists a polynomial time algorithm that takes as input a free-connex CQ* $Q(\vec{x}, \star, \vec{z})$ without self-joins and a relation symbol R of Q, and produces a* full *acyclic CQ* $Q'(\vec{x}, \star, \vec{z})$ without self-joins and a relation symbol R' of Q', so that the following are equivalent:*

1. *Direct access for Q over R-annotated $\mathbb{K}$-databases is in $\langle \text{loglinear}, \log \rangle$.*
2. *Direct access for Q' over R'-annotated $\mathbb{K}$-databases is in $\langle \text{loglinear}, \log \rangle$.*

From Lemma 19 we conclude that, to determine the tractability of the CQ* Q, it suffices to determine the tractability of the CQ* Q', which has the property of being full. Using Lemma 19 and Theorem 17, we can now prove our classification of the CQ*s without self-joins over databases locally annotated in the case of an idempotent addition.

▶ **Theorem 20.** *Let $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ be a logarithmic-time $\oplus$-idempotent commutative semiring. Let R be a relation symbol of a free-connex CQ* Q without self-joins. Let Q' and R' be the CQ* and relation symbol obtained from Q and R using Lemma 19, and let Q_0 be the R'-deanotation of Q'.*

1. *If Q_0 has no disruptive trio, then direct access for Q over R-annotated $\mathbb{K}$-databases is in $\langle \text{loglinear}, \log \rangle$.*
2. *Otherwise, in the case where $\mathbb{N} \subseteq \mathbb{K}$, direct access for Q over R-annotated $\mathbb{K}$-databases is not in $\langle \text{loglinear}, \log \rangle$, assuming the SparseBMM hypothesis.*

Proof. The proof argues about three queries:

1. The input CQ* $Q(\vec{x}, \star, \vec{z})$;
2. A full acyclic CQ* $Q'(\vec{x}, \star, \vec{z})$;

3. A full acyclic CQ Q_0.

Given Q and R as described in Theorem 20, we eliminate existential variables using Lemma 19 and obtain Q' and a relation symbol R'. The CQ Q_0 is the R'-deannotation of Q'. We can show that Q_0 is acyclic since Q' is acyclic (and for the exact proof, see the full version of the paper [10]).

From Lemma 19 we conclude that direct access for Q over R-annotated $\mathbb{K}$-databases is in $\langle \text{loglinear}, \log \rangle$ if and only if direct access for Q' over R'-annotated $\mathbb{K}$-databases is in $\langle \text{loglinear}, \log \rangle$. So, it remains to prove that direct access for Q' over R'-annotated $\mathbb{K}$-databases is in $\langle \text{loglinear}, \log \rangle$ if and only if Q_0 has no disruptive trio.

Suppose first that Q_0 has no disruptive trio. Part 1 of Theorem 17 implies that direct access for Q' over R'-annotated $\mathbb{K}$-databases is in $\langle \text{loglinear}, \log \rangle$, and therefore, so is direct access for Q over R-annotated $\mathbb{K}$-databases. Conversely, suppose that Q_0 has a disruptive trio. Suppose also that $\mathbb{N} \subseteq \mathbb{K}$, as we assume in Item 2 of Theorem 20. Then Part 2 of Theorem 17 states that direct access for Q' over R'-annotated $\mathbb{K}$-databases is not in $\langle \text{loglinear}, \log \rangle$, assuming SparseBMM. This completes the proof. ◀

Note that it follows from Theorem 20 that, when $\mathbb{N} \subseteq \mathbb{K}$, one can determine in polynomial time whether a given CQ* is in $\langle \text{loglinear}, \log \rangle$ or not, assuming the SparseBMM hypothesis. Also, as in Corollary 6, we can directly reason about ACQs using Theorem 20. Consider a free-connex ACQ $Q(\vec{x}, \alpha(\vec{y}), \vec{z}) :- \varphi_1(\vec{x}, \vec{y}), \ldots, \varphi_\ell(\vec{x}, \vec{y})$, where α is one of CountD over a logarithmic domain, Min, or Max. We can transform the ACQ Q into a CQ* Q', and then apply Theorem 20 in order to transform Q' into a CQ Q_0. If Q_0 has no disruptive trio, then we get direct access for Q in $\langle \text{loglinear}, \log \rangle$. So, at the end of the day, we reduce an ACQ into a CQ*, which is transformed into a full CQ* that, in turn, is transformed into a CQ that we solve in $\langle \text{loglinear}, \log \rangle$.

Section summary. Theorem 12 gives a sufficient condition for tractability of CQ*s over any logarithmic-time commutative semiring. When inspecting the cases not covered by this sufficient condition, Theorem 9 shows that even simple queries may introduce hardness. Therefore, we narrow our focus to the form of annotation obtained from ACQs (as defined here): locally annotated databases. Theorem 17 shows a dichotomy for full CQ*s without self-joins. Beyond full CQ*s, we notice that there are cases, like that of Theorem 10, where the hardness can be attributed to the semiring. We identify the class of $\oplus$-idempotent semirings as one that facilitates direct access. In the context of such semirings, Theorem 20 extends the dichotomy of Theorem 17 to support existential variables.

6 Concluding Remarks

Direct access provides an opportunity to efficiently evaluate queries even when the number of answers is enormous. Past research studied the feasibility of direct access for CQs, and here we embarked on the exploration of this problem for queries that involve aggregation, either as standard (SQL) aggregate functions or annotation with commutative semirings, that is, CQ*s and ACQs. We studied the challenges of *construction* and *ordering by* the computed value (aggregation or annotation). We showed that past results carry over to include the computed value, and particularly, that the past classification holds as long as the computed value is not involved in the order. For the second challenge, involving the computed value in the order introduces hardness pretty quickly. We showed a sufficient condition that allows incorporating the computed value in a nontrivial manner. Moreover, we established a full

classification of the complexity of CQ*s without self-joins in the case of databases locally annotated by a semiring with an idempotent addition.

An important direction for future work is the exploration of queries beyond free-connex ones; for that, we need to allow for broader yardsticks of efficiency, as done for direct access for CQs without aggregation [5] and as done for Functional Aggregate Queries (FAQ) for CQs with aggregation and traditional query evaluation [16]. Moreover, we plan to explore the extension of our results to queries with self-joins, and we believe that recent results [5] can be used towards such an extension. It is also left for future work to better understand the limits of computation and establish lower bounds (and dichotomies) for general classes of queries, commutative semirings, and aggregate functions. Another important direction is to explore the ability to efficiently maintain the direct-access structure through updates of the database, as previously studied in the context of non-aggregate queries [3, 21]. Finally, we plan to investigate the practical behavior of our algorithms and understand how well the theoretical acceleration is realized in comparison to existing query engines.

References

1 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic*, pages 208–222, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

2 Guillaume Bagan, Arnaud Durand, Etienne Grandjean, and Frédéric Olive. Computing the JTH solution of a first-order query. *RAIRO - Theoretical Informatics and Applications (RAIRO: ITA)*, 42:147–164, 2008. URL: `https://hal.archives-ouvertes.fr/hal-00221730`.

3 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering FO+MOD queries under updates on bounded degree databases. *ACM Trans. Database Syst.*, 43(2):7:1–7:32, 2018.

4 Johann Brault-Baron. *De la pertinence de l'énumération : complexité en logiques propositionnelle et du premier ordre*. Theses, Université de Caen, April 2013.

5 Karl Bringmann, Nofar Carmeli, and Stefan Mengel. Tight fine-grained bounds for direct access on join queries. In *PODS*, pages 427–436. ACM, 2022.

6 Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. Tractable orders for direct access to ranked answers of conjunctive queries. In *PODS*, pages 325–341. ACM, 2021.

7 Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Alessio Conte, Benny Kimelfeld, and Nicole Schweikardt. Answering (unions of) conjunctive queries using random access and random-order enumeration. *ACM Trans. Database Syst.*, 47(3):9:1–9:49, 2022.

8 Henri Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate texts in mathematics*. Springer, 1993.

9 Sara Cohen, Werner Nutt, and Yehoshua Sagiv. Deciding equivalences among conjunctive aggregate queries. *J. ACM*, 54(2):5–es, apr 2007.

10 Idan Eldar, Nofar Carmeli, and Benny Kimelfeld. Direct access for answers to conjunctive queries with aggregation. *CoRR*, abs/2303.05327, 2023.

11 Anka Gajentaan and Mark H Overmars. On a class of o(n2) problems in computational geometry. *Computational Geometry*, 5(3):165–185, 1995. URL: `https://www.sciencedirect.com/science/article/pii/0925772195000222`, `doi:https://doi.org/10.1016/0925-7721(95)00022-2`.

12 Étienne Grandjean and Louis Jachiet. Which arithmetic operations can be performed in constant time in the ram model with addition?, 2022. URL: `https://arxiv.org/abs/2206.13851`, `doi:10.48550/ARXIV.2206.13851`.

13 Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, PODS '07, page 31–40, New York, NY, USA, 2007. Association for Computing Machinery.

14 Muhammad Idris, Martin Ugarte, and Stijn Vansummeren. The dynamic yannakakis algorithm: Compact and efficient query processing under updates. In *SIGMOD*, page 1259–1274, New York, NY, USA, 2017. Association for Computing Machinery.

15 Mahmoud Abo Khamis, Ryan R. Curtin, Benjamin Moseley, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Functional aggregate queries with additive inequalities. *ACM Trans. Database Syst.*, 45(4):17:1–17:41, 2020.

16 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. In *PODS*, pages 13–28. ACM, 2016.

17 Dan Olteanu and Maximilian Schleich. Factorized databases. *SIGMOD Rec.*, 45(2):5–16, 2016.

18 Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Trans. Database Syst.*, 40(1), mar 2015.

19 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC '10, page 603–610, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1806689.1806772.

20 Christopher Ré and Dan Suciu. The trichotomy of HAVING queries on a probabilistic database. *VLDB J.*, 18(5):1091–1116, 2009.

21 Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Dynamic complexity under definable changes. *ACM Trans. Database Syst.*, 43(3):12:1–12:38, 2018.

22 Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *IPEC*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

23 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, VLDB '81, page 82–94. VLDB Endowment, 1981.