Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries

NOFAR CARMELI, Technion, Israel and DI ENS, ENS, CNRS, PSL University, Inria, France
NIKOLAOS TZIAVELIS and WOLFGANG GATTERBAUER, Northeastern University, USA
BENNY KIMELFELD, Technion - Israel Institute of Technology, Israel
MIREK RIEDEWALD, Northeastern University, USA

We study the question of when we can provide *direct access to the k-th answer* to a Conjunctive Query (CQ) according to a specified order over the answers in time logarithmic in the size of the database, following a preprocessing step that constructs a data structure in time quasilinear in database size. Specifically, we embark on the challenge of identifying *the tractable answer orderings*, that is, those orders that allow for such complexity guarantees. To better understand the computational challenge at hand, we also investigate the more modest task of providing access to only a single answer (i.e., finding the answer at a given position), a task that we refer to as *the selection problem*, and ask when it can be performed in quasilinear time. We also explore the question of when selection is indeed easier than ranked direct access.

We begin with *lexicographic orders*. For each of the two problems, we give a decidable characterization (under conventional complexity assumptions) of the class of tractable lexicographic orders for every CQ without self-joins. We then continue to the more general *orders by the sum of attribute weights* and establish the corresponding decidable characterizations, for each of the two problems, of the tractable CQs without self-joins. Finally, we explore the question of when the satisfaction of Functional Dependencies (FDs) can be utilized for tractability and establish the corresponding generalizations of our characterizations for every set of unary FDs.

CCS Concepts: • **Theory of computation** → **Database theory**; *Complexity classes*; *Database query languages (principles)*; *Database query processing and optimization (theory);*

Additional Key Words and Phrases: Conjunctive queries, direct access, ranking function, answer orderings, query classification

N. Carmeli and N. Tziavelis contributed equally to the article.

Nofar Carmeli and Nikolaos Tziavelis were supported by Google PhD Fellowships. Nofar Carmeli and Benny Kimelfeld were supported by the German Research Foundation (DFG) Project 412400621 (DIP program). Nikolaos Tziavelis, Wolfgang Gatterbauer, and Mirek Riedewald were supported by the National Science Foundation (NSF) under award number IIS-1956096. Wolfgang Gatterbauer was supported by NSF under award number CAREER IIS-1762268. Nofar Carmeli was supported by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute).

Authors' addresses: N. Carmeli, DI ENS, 45 rue d'Ulm, 75005 Paris, France; email: Nofar.Carmeli@inria.fr; N. Tziavelis, W. Gatterbauer, and M. Riedewald, Northeastern University, Khoury College of Computer Sciences, 440 Huntington Avenue, 442 West Village H, Boston, MA, USA, 02115; emails: {tziavelis.n, w.gatterbauer, m.riedewald}@northeastern.edu; B. Kimelfeld, The Taub Faculty of Computer Science, Technion - Israel Institute of Technology, Haifa 3200003, Israel; email: bennyk@cs.technion.ac.il.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

0362-5915/2023/03-ART1 $15.00
https://doi.org/10.1145/3578517

ACM Reference format:
Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. 2023. Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries. *ACM Trans. Datab. Syst.* 48, 1, Article 1 (March 2023), 45 pages.
https://doi.org/10.1145/3578517

1 INTRODUCTION

When can we support direct access to a ranked list of answers to a database query without (and considerably faster than) materializing all answers? To illustrate the concrete instantiation of this question, assume the following simple relational schema for information about activities of residents in the context of pandemic spread:

$$\text{Visits}(\text{person}, \text{age}, \text{city}) \qquad \text{Cases}(\text{city}, \text{date}, \#\text{cases}).$$

Here, the relation Visits mentions, for each person in the database, the cities that the person visits regularly (e.g., for work and for visiting relatives) and the age of the person (for risk assessment); the relation Cases specifies the number of new infection cases in specific cities at specific dates (a measure that is commonly used for spread assessment albeit being sensitive to the amount of testing).

Suppose that we wish to efficiently compute the natural join Visits $\bowtie$ Cases based on equality of the city attribute, so we have all combinations of people (with their age), the cities they regularly visit, and the city's daily new cases. For example,

$$(\texttt{Anna}, 72, \texttt{Boston}, \texttt{12/7/2020}, 179)\,.$$

While the number of such answers could be quadratic in the size of the database, the seminal work of Bagan, Durand, and Grandjean [4] has established that the answers can be enumerated with a constant delay between consecutive answers after a linear-time preprocessing phase. This is due to the fact that this join is a special case of a *free-connex* Conjunctive Query (CQ). In the case of CQs without self-joins, being free-connex is a sufficient and necessary condition for such efficient evaluation [4, 11]. The necessity requires conventional assumptions in fine-grained complexity[1] and it holds even if we multiply the preprocessing time and delay by a logarithmic factor in the size of the database.[2]

To realize the constant (or logarithmic) delay, the preprocessing phase builds a data structure that enables efficient iteration over the answers in the enumeration phase. Brault-Baron [11] showed that in the quasilinear-time preprocessing phase, we can build a structure with better guarantees: not only log-delay enumeration, but even log-time *direct access*: a structure that, given k, allows to directly retrieve the k-th answer in the enumeration without needing to enumerate the preceding $k-1$ answers first.[3] Later, Carmeli et al. [14] showed how such a structure can be used for enumerating answers in a random order (random permutation)[4] with the statistical guarantee that the order is uniformly distributed. In particular, in the above example, we can enumerate the answers of Visits $\bowtie$ Cases in a provably uniform random permutation (hence, ensuring statistical validity of each prefix) with logarithmic delay, after a quasilinear-time preprocessing phase.

[1] For the sake of simplicity, throughout this section, we make all of these complexity assumptions. We give their formal statements in Section 2.4.
[2] We refer to those as *quasilinear preprocessing* and *logarithmic delay*, respectively.
[3] "Direct access" is also widely known as "random access." We prefer to use "direct access" to avoid confusion with the problem of answering "in random order."
[4] Not to be confused with "random access."

Their direct-access structure also allows for *inverted access*: Given an answer, return the index k of that answer (or determine that it is not a valid answer). Recently, Keppeler [31] proposed another direct-access structure with the additional ability to allow efficient database updates, but at the cost of only supporting a limited subset of free-connex CQs.

All known direct-access structures [11, 14, 31] allow the answers to be *sorted* by some lexicographic order (even if the formal results do not explicitly state it). For instance, in our example of Visits $\bowtie$ Cases, the structure could be such that the tuples are enumerated in the (descending or ascending) order of #cases and then by date, or in the order of city and then by age. Hence, in logarithmic time, we can evaluate quantile queries, namely, find the k-th answer in order and determine the position of a tuple inside the sorted list. From this, we can also conclude (fairly easily) that we can enumerate the answers ordered by age where ties are broken randomly, again provably uniformly. Carmeli et al. [14] have also shown how the order of the answers can be useful for generalizing direct-access algorithms from CQs to UCQs. Notice that direct access to the sorted list of answers is a stronger requirement than *ranked enumeration* that has been studied in recent work [10, 15, 40, 41, 43, 45].

Yet, the choice of lexicographic order is an artefact of the structure construction, e.g., the elimination order [11], the join tree [14], or the q-tree [8]. If the application desires a specific lexicographic order, then we can only hope to find a matching construction. However, this is not necessarily possible. For example, could we construct in quasilinear time a direct-access structure for Visits $\bowtie$ Cases ordered by #cases and then by age? Interestingly, we will show that the answer is negative: It is impossible to build in quasilinear time a direct-access structure with logarithmic access time for that lexicographic order.

Getting back to the question posed at the beginning of this section, in this article, we embark on the challenge of identifying, for each CQ, the orders that allow for efficiently constructing a direct-access structure. We adopt the *tractability yardstick* of quasilinear construction (preprocessing) time and logarithmic access time. In addition, we focus on two types of orders: lexicographic orders and scoring by the sum of attribute weights.

As aforesaid, some of the orders that we study are intractable. To understand the root cause of the hardness, we consider another task that allows us to narrow our question to a considerably weaker guarantee. Our notion of tractability so far requires the construction of a structure in quasilinear time that allows direct access in logarithmic time. In particular, if our goal is to compute just a *single* quantile, say the k-th answer, then it should take quasilinear time. Computing a single quantile is known as the *selection problem* [9]. The question we ask is to what extent selection is a weaker requirement than direct access In other words, do we get more tractable cases if we lift the requirement to construct a data structure and instead ask for quasilinear time per access?

In some situations, we might be able to avoid hardness through a more careful inspection of the *integrity constraints* that the database guarantees on the source relations. For example, it turns out that we *can* construct in quasilinear time a direct-access structure for Visits $\bowtie$ Cases ordered by #cases and then by age if we assume that each city occurs at most once in Cases (i.e., for each city, we have a report for a single day). Hence, it may be the case that an ordered CQ is classified as intractable (with respect to the guarantees that we seek), but it becomes tractable if we are allowed to assume that the input database satisfies some integrity constraints such as key constraints or more general Functional Dependencies (FDs). Moreover, FDs are so common that ignoring them implies that we often miss opportunities of fast algorithms. This angle arises regardless of answer ordering, and indeed, Carmeli and Kröll [12] showed precisely how the class of (self-join-free) CQs with tractable enumeration extends in the presence of FDs. Accordingly, we extend our study on ranked direct access and the selection problem to incorporate FDs, and aim to classify every combination of *(a)* CQ, *(b)* order over the answers, *and (c)* set of FDs.

Contributions. Before we describe the results that we establish in this manuscript, let us illustrate them on an example.

Example 1.1. Figure 1 depicts an example database and different orderings of the answers to the 2-path CQ $Q(x, y, z) :- R(x, y), S(y, z)$. The question we ask is whether the median (e.g., the third answer in the example) or in general, the answer in any index can be computed efficiently as the database size n grows. Tractable direct access requires $O(\text{polylog } n)$ per access after $O(n \text{ polylog } n)$ preprocessing, while tractable selection requires $O(n \text{ polylog } n)$ for a single access. We compare the impact of different orders, projections, and FDs on the example 2-path CQ.

- LEX $\langle x, y, z\rangle$: Direct access is tractable.
- LEX $\langle x, z, y\rangle$: Direct access is intractable, because the lexicographic order "does not agree" with the query structure, which we capture through the concept of a *disruptive trio* that we introduce later on. However, selection is tractable.
- LEX $\langle x, z\rangle$: Direct access is intractable, because the variables in the partial lexicographic order are not "connected" in a particular way. We will define this as not being L-connex for a lexicographic order L. However, selection is again tractable.
- LEX $\langle x, z\rangle$ and y projected away: Selection is now intractable, because the query is not free-connex.
- LEX $\langle x, z, y\rangle$, with the FD $R : y \rightarrow x$ or the FD $S : y \rightarrow z$: Direct access is tractable as a consequence of earlier work on enumeration with FDs [12].
- LEX $\langle x, z, y\rangle$, with the FD $R : x \rightarrow y$: Direct access is tractable with the techniques that we develop in this article. Intuitively, the FD implies that the order is equivalent to the tractable order $\langle x, y, z\rangle$.
- LEX $\langle x, z, y\rangle$, with the FD $S : z \rightarrow y$: Direct access is intractable, since the FD does not help in this case.
- SUM $x + y + z$: Direct access is intractable, because it would allow us to solve the 3SUM problem in subquadratic time, yet selection is tractable.
- SUM $x + y$ and z projected away: Direct access is tractable, because all the free variables are contained in R. This means that we can produce the sorted list of answers during preprocessing.
- SUM $x + z$ and y projected away: Selection is intractable, because the query is not free-connex.

(1) Our first main result is an algorithm for direct access for lexicographic orders, including ones that are not achievable by past structures. We further show that within the class of CQs without self-joins, our algorithm covers all the tractable cases (in the sense adopted here), and we establish a decidable and easy-to-test classification of the lexicographic orders over the free variables into tractable and intractable ones. For instance, in the case of Visits $\bowtie$ Cases the lexicographic order (#cases, age, city, date, person) is intractable. It is classified as such because #cases and age are non-neighbors (i.e., do not co-occur in the same atom), but city, which comes after #cases and age in the order, is a neighbor of both. This is what we call a *disruptive trio*. The lexicographic order (#cases, age) is also intractable, since the query Visits $\bowtie$ Cases is not {#cases, age}-connex (a similar condition to being free-connex, but for the subset of the variables that appear in the lexicographic order instead of the free ones). In contrast, the lexicographic order (#cases, city, age) is tractable. We also show that, within the tractable side, the structure we construct allows for inverted access in constant time.

Our classification is proved in two steps. We begin by considering the complete lexicographic orders (that involve all free variables). We show that for free-connex CQs without self-joins, the absence of a disruptive trio is a sufficient and necessary condition for tractability. We then

R	x	y
	1	5
	1	2
	6	2

S	y	z
	5	3
	5	4
	5	6
	2	8

(a) Example Database.

Q	x	y	z
#1	1	2	8
#2	1	5	3
#3	1	5	4
#4	1	5	6
#5	6	2	8

(b) LEX ordering $\langle x, y, z\rangle$.

Q	x	z	y
#1	1	3	5
#2	1	4	5
#3	1	6	5
#4	1	8	2
#5	6	8	2

(c) LEX ordering $\langle x, z, y\rangle$

Q	x	y	z	$x+y+z$
#1	1	5	3	9
#2	1	5	4	10
#3	1	2	8	11
#4	1	5	6	12
#5	6	2	8	16

(d) SUM ordering.

Fig. 1. Example 1.1: An example input database (a) and possible orderings of the answers to the query $Q(x, y, z) :- R(x, y), S(y, z)$. The orderings in (b) and (c) use two different lexicographic orders (LEX), while the ordering in (d) uses a sum-of-weights order where the weights are assumed to be identical to the attribute values.

generalize to partial lexicographic orders L where the ordering is determined only by a *subset of the free variables*. There, the condition is that there is no disruptive trio *and* that the query is L-connex. Interestingly, it turns out that a partial lexicographic order is tractable if and only if it is the prefix of a complete tractable lexicographic order.

(2) Next, we study the selection problem for lexicographic orders and show that being free-connex is a sufficient and necessary condition for a linear-time solution in the case of CQs without self-joins. In particular, there are ordered queries with tractable selection but intractable direct access, namely, free-connex CQs without self-joins where we have a disruptive trio or lack the property of being L-connex.

(3) A lexicographic order is a special case of an ordering by the *sum of attribute weights*, where every database value is mapped to some weight. Hence, a natural question is which CQs have a tractable direct access by the order of sum. For example, what about Visits $\bowtie$ Cases with the order (α·#cases + β·age)? It is easy to see that this order is intractable because the lexicographic order (#cases, age) is intractable. In fact, it is easy to show that an order by sum is intractable whenever *there exists* an intractable lexicographic order (e.g., there is a disruptive trio). However, we will show that the situation is worse: The only tractable case is the one where the CQ is free-connex and there is an atom that contains all of the free variables. In particular, ranked direct access by sum is intractable already for the Cartesian product $Q(c_1, d, x, p, a, c_2) :- \text{Visits}(p, a, c_1), \text{Cases}(c_2, d, x)$, even though *every* lexicographic order is tractable (according to our aforementioned classification). This daunting hardness also emphasizes how ranked direct access is fundamentally harder than *ranked enumeration* where, in the case of the sum of attributes, the answers of *every free-connex* CQ can be enumerated with logarithmic delay after a linear preprocessing time [43].

(4) Next, we study the selection problem for the sum of weights and establish the following dichotomy in complexity (again assuming fine-grained hypotheses): The selection problem can be solved in $O(n \log n)$ time, where n is the size of the database, if and only if the hypergraph of the CQ restricted to the free variables contains at most two maximal hyperedges (w.r.t. containment). The tractable side is applicable even in the presence of self-joins, and it is achieved by adopting an algorithm by Frederickson and Johnson [20] originally developed for selection on sorted matrices. For illustration, the selection problem is solvable in quasilinear time for the query Visits $\bowtie$ Cases ordered by sum.

(5) Last, we study the implication of FDs on our results and generalize all of them to incorporate a set of unary FDs (i.e., FDs with a single attribute on the left-hand side). Like previous works on FDs on enumeration [12], deletion propagation [32], resilience [21], and probabilistic inference [23], we use the notion of an extended CQ to reason about the tractability of a CQ under the presence of

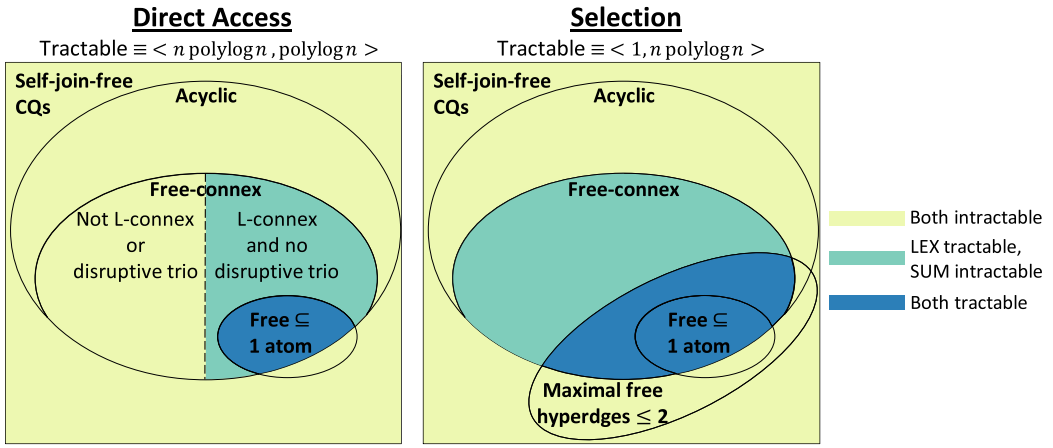


Fig. 2. Overview of our results for lexicographic (LEX) orders and sum-of-weights (SUM) orders. CQs without self-joins (SJ-free) are classified based on the tractability of the *direct access problem* (left) and the *selection problem* (right). The L-connex property applies only to lexicographic orders L (the precise definitions are given in Section 2). All tractable cases extend to CQs with self-joins. The sizes of the ellipses are arbitrary and do not correspond to the size or importance of the classes.

FDs. The idea is that by looking at the structure of the extended CQ (without FDs), then we are able to classify the original CQ together with the FDs. While this works in a relatively straightforward way for the case of sum, the case of lexicographic orders is more involved, since the FDs may interact with the order in non-trivial ways. To extend our dichotomy results for lexicographic orders to incorporate FDs, we show how the extension of a CQ and order may also result in a *reordering* of the variables. Then, tractability is decided by the extended CQ together with the reordered lexicographic order.

Overview of results. We summarize our results (excluding the dichotomies under the presence of FDs) in Figure 2 with different colors indicating the tractability of the studied orders, namely, lexicographic (LEX) and sum-of-weights (SUM) orders. For both direct access and selection, we obtain the precise picture of the orders and CQs without self-joins that are tractable according to our yardstick: $O(n \operatorname{polylog} n)$ preprocessing and $O(\operatorname{polylog} n)$ per access for direct access (conveniently denoted as $\langle n \operatorname{polylog} n, \operatorname{polylog} n\rangle$ for $\langle$preprocessing, access$\rangle$) and $O(n \operatorname{polylog} n)$ for selection (or $\langle 1, n \log n\rangle$). Finally, we show how the results are affected by every set of unary FDs (not depicted in Figure 2); in other words, we extend our dichotomies to incorporate the FDs of the underlying schema under the restriction that all FDs have a single attribute on the premise. We leave the case of more general FDs open for future research.

Comparison to an earlier conference version. A preliminary version of this manuscript appeared in a conference proceedings [13]. Compared to that version, this manuscript includes significant extensions and improvements. First, we added an investigation on the complexity of the selection problem with lexicographic orders, establishing a complete dichotomy result (Theorem 6.1 and all of Section 6). Second, we extended a dichotomy from the conference version to include self-join-free CQs *beyond full CQs* for the selection problem by SUM, so it now covers all self-join-free CQs (with projections), thereby resolving the corresponding open question from the conference paper (Section 7.4). Third, we extended our results to cover unary FDs (Section 8). Fourth, we have clarified the relationship between the disruptive trio and the concept of an elimination order

(Remark 1). Fifth and last, we made considerable simplifications and improvements in previous components, including the proof of hardness of direct access for lexicographic orders (Lemma 3.13) and several proofs for the SUM selection (Section 7).

Applicability. It is important to note that while our positive results are stated over a limited class of queries (a fragment of acyclic CQs), there are some implications beyond this class that are immediate yet significant. In particular, we can use known techniques that reduce other CQs to a tractable form and then apply our direct-access or selection algorithms. As an example, a *hypertree decomposition* can be used to transform a cyclic CQ to an acyclic form by paying a non-linear overhead during preprocessing [26]. As another example, a CQ with inequality ($<$) predicates can be reduced to a CQ without inequalities by paying only a polylogarithmic-factor increase in the size of the database [42].

Outline. The remainder of the manuscript is organized as follows: Section 2 gives the necessary background. In Section 3, we consider direct access by lexicographic orders that include all the free variables, and Section 4 extends the results to partial ones. We move on to the (for the most part) negative results for direct access by sum orders in Section 5. We study the selection problem for lexicographic orders and sum in Sections 6 and 7, respectively. We extend our results to incorporate unary FDs in Section 8 and, last, conclude and give future directions in Section 9.

2 PRELIMINARIES

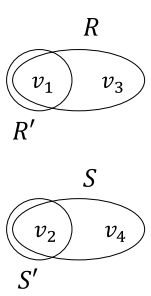
2.1 Basic Notions

Database. A *schema* $\mathcal{S}$ is a set of relational symbols $\{R_1, \ldots, R_m\}$. We use $\mathtt{ar}(R)$ for the arity of a relational symbol R. A *database instance* I contains a finite relation $R^I \subseteq \mathtt{dom}^{\mathtt{ar}(R)}$ for each $R \in \mathcal{S}$, where $\mathtt{dom}$ is a set of constant values called the *domain*. When I is clear, we simply use R instead of R^I. We use n for the size of the database, i.e., the total number of tuples.

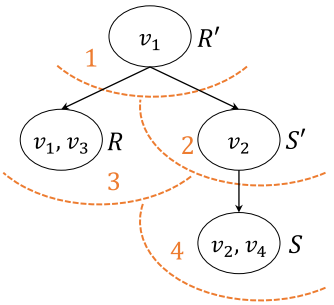
Queries. A *conjunctive query* (CQ) Q over schema $\mathcal{S}$ is an expression of the form $Q(\mathbf{X}_f) \coloneq R_1(\mathbf{X}_1), \ldots, R_\ell(\mathbf{X}_\ell)$, where the tuples $\mathbf{X}_f, \mathbf{X}_1, \ldots, \mathbf{X}_\ell$ hold variables, every variable in $\mathbf{X}_f$ appears in some $\mathbf{X}_1, \ldots, \mathbf{X}_\ell$, and $\{R_1, \ldots, R_\ell\} \subseteq \mathcal{S}$. Each $R_i(\mathbf{X}_i)$ is called an *atom* of the query Q, and the set of all atoms is denoted by $\mathtt{atoms}(Q)$. We use $\mathtt{var}(e)$ or $\mathtt{var}(Q)$ for the set of variables that appear in an atom e or query Q, respectively.[5] The variables $\mathbf{X}_f$ are called *free* and are denoted by $\mathtt{free}(Q)$. A CQ is *full* if $\mathtt{free}(Q) = \mathtt{var}(Q)$ and *Boolean* if $\mathtt{free}(Q) = \emptyset$. Sometimes, we say that CQs that are not full have *projections*. A repeated occurrence of a relational symbol is a *self-join* and if no self-joins exist, a CQ is called *self-join-free*. A homomorphism from a CQ Q to a database I is a mapping of $\mathtt{var}(Q)$ to constants from $\mathtt{dom}$, such that every atom of Q maps to a tuple in the database I. A *query answer* is such a homomorphism followed by a projection on the free variables. The answer to a Boolean CQ is whether such a homomorphism exists. The set of query answers is $Q(I)$ and we use $q \in Q(I)$ for a query answer. For an atom $R(\mathbf{X})$ of a CQ, we say that a tuple $t \in R$ assigns a value c to a variable x and denote it as $t[x] = c$ if for every index i such that $\mathbf{X}[i] = x$, we have that $t[i] = c$. The *active domain* of a variable x is the subset of $\mathtt{dom}$ that x can be assigned from the database I.

Hypergraphs. A *hypergraph* $\mathcal{H} = (V, E)$ is a set V of *vertices* and a set E of subsets of V called *hyperedges*. Two vertices in a hypergraph are *neighbors* if they appear in the same edge. A *path* of $\mathcal{H}$ is a sequence of vertices such that every two succeeding vertices are neighbors. A *chordless path* is a path in which no two non-succeeding vertices appear in the same hyperedge (in particular, no vertex appears twice). A *join tree* of a hypergraph $\mathcal{H} = (V, E)$ is a tree T where

[5] We use e for atoms because of the natural analogy to hyperedges in hypergraphs associated with a query Q.



(a) A hypergraph that is inclusion equivalent to $\mathcal{H}(Q_3)$.



(b) A layered join tree for Q_3 w.r.t. the lexicographic order.

Fig. 3. Constructing a layered join tree for the query $Q_3(v_1, v_2, v_3, v_4) :- R(v_1, v_3), S(v_2, v_4)$ and order $\langle v_1, v_2, v_3, v_4 \rangle$.

the nodes[6] are the hyperedges of $\mathcal{H}$ and the *running intersection* property holds, namely: For all $u \in V$ the set $\{e \in E \mid u \in e\}$ forms a (connected) subtree in T. An equivalent phrasing of the running intersection property is that given two nodes e_1, e_2 of the tree, for any node e_3 on the simple path between them, we have that $e_1 \cap e_2 \subseteq e_3$. A hypergraph $\mathcal{H}$ is *acyclic* if there exists a join tree for $\mathcal{H}$. We associate a hypergraph $\mathcal{H}(Q) = (V, E)$ to a CQ Q where the vertices are the variables of Q, and every atom of Q corresponds to a hyperedge with the same set of variables. Stated differently, $V = \mathtt{var}(Q)$ and $E = \{\mathtt{var}(e) | e \in \mathtt{atoms}(Q)\}$. With a slight abuse of notation, we identify atoms of Q with hyperedges of $\mathcal{H}(Q)$. A CQ Q is *acyclic* if $\mathcal{H}(Q)$ is acyclic, otherwise it is *cyclic*. The free-restricted hypergraph $\mathcal{H}_{\mathtt{free}}(Q)$ is the restriction of $\mathcal{H}(Q) = (V, E)$ on the nodes that correspond to free variables, i.e., $\mathcal{H}_{\mathtt{free}}(Q) = (\mathtt{free}(Q), \{e \cap \mathtt{free}(Q) | e \in E\})$.

Free-connex CQs. A hypergraph $\mathcal{H}'$ is an *inclusive extension* of $\mathcal{H}$ if every edge of $\mathcal{H}$ appears in $\mathcal{H}'$, and every edge of $\mathcal{H}'$ is a subset of some edge in $\mathcal{H}$. Given a subset $\mathbf{X}$ of the vertices of $\mathcal{H}$, a tree T is an *ext-$\mathbf{X}$-connex tree* (i.e., extension-$\mathbf{X}$-connex tree) for a hypergraph $\mathcal{H}$ if: (1) T is a join tree of an inclusive extension of $\mathcal{H}$ and (2) there is a subtree[7] T' of T that contains exactly the vertices $\mathbf{X}$ [4]. As an example, Figure 3(b) depicts an ext-$\{v_1, v_2\}$-connex tree for a query Q_3. We say that a hypergraph is $\mathbf{X}$-connex if it has an ext-$\mathbf{X}$-connex tree [4]. A hypergraph is $\mathbf{X}$-connex iff it is acyclic and it remains acyclic after the addition of a hyperedge containing exactly $\mathbf{X}$ [7, 11]. Given a hypergraph $\mathcal{H}$ and a subset $\mathbf{X}$ of its vertices, an *$\mathbf{X}$-path* is a chordless path $(x, z_1, \ldots, z_k, y)$ in $\mathcal{H}$ with $k \geq 1$, such that $x, y \in \mathbf{X}$, and $z_1, \ldots, z_k \notin \mathbf{X}$. A hypergraph is $\mathbf{X}$-connex iff it has no $\mathbf{X}$-path [4]. A CQ Q is *free-connex* if $\mathcal{H}(Q)$ is $\mathtt{free}(Q)$-connex [4]. Note that a free-connex CQ is necessarily acyclic.[8] An implication of the characterization given above is that it suffices to find a join-tree for an inclusive extension of a hypergraph $\mathcal{H}$ to infer that $\mathcal{H}$ is acyclic.

To simplify notation, we also say that a CQ is L-connex for a (partial) lexicographic order L if the CQ is $\mathbf{X}$-connex for the set of variables $\mathbf{X}$ that appear in L. Generalizing the notion of an inclusive extension, we say that a hypergraph $\mathcal{H}'$ is *inclusion-equivalent* to $\mathcal{H}$ if every hyperedge of $\mathcal{H}$ is a subset of some hyperedge of $\mathcal{H}'$ and vice versa. For example, the two hypergraphs with hyperedges $E_1 = \{\{x, y\}, \{y, z\}\}$ and $E_2 = \{\{x, y\}, \{y, z\}, \{z\}\}$ are inclusion-equivalent, because $\{z\}$ is a subset of $\{y, z\}$ and every hyperedge is trivially a subset of itself.

[6] To make a clear distinction between the vertices of a hypergraph and those of its join tree, we call the latter nodes.
[7] By subtree, we mean any connected subgraph of the tree.
[8] Free-connex CQs are sometimes called in the literature *free-connex acyclic* CQs [4]. For cyclic CQs, the free-connex property is only defined with respect to a particular hypertree decomposition [4] and not for the CQs themselves. Thus, we choose to omit the word *acyclic* when referring to free-connex acyclic CQs.

2.2 Problem Definitions

Orders over Answers. For a CQ Q and database instance I, we assume a *total order* $\preceq$ over the query answers $Q(I)$. We consider two types of orders in this article:

(1) **LEX:** Assuming that the domain values are ordered, a *lexicographic order* L is an ordering of $\texttt{free}(Q)$ such that $\preceq$ compares two query answers q_1, q_2 on the value of the first variable in L, then on the second (if they are equal on the first), and so on [28]. A lexicographic order is called *partial* if the variables in L are a subset of $\texttt{free}(Q)$.

(2) **SUM:** The second type of order assumes given weight functions that assign real-valued weights to the domain values of each variable. More precisely, for each variable x, we define a function $w_x : \texttt{dom} \to \mathbb{R}$. Then, the query answers are ordered by a weight that is computed by aggregating the weights of the assigned values of free variables. In a *sum-of-weights order*, denoted by SUM, we have the weight of each query answer $q \in Q(I)$ to be $w_Q(q) = \sum_{x \in \texttt{free}(Q)} w_x(q(x))$ and $q_1 \preceq q_2$ implies that $w_Q(q_1) \leq w_Q(q_2)$. We emphasize that we allow only free variables to have weights, otherwise different semantics for the query answers are possible [43]. To simplify notation, we sometimes refer to all $w_x, x \in \texttt{free}(Q)$ and w_Q together as one weight function w.

Attribute Weights vs. Tuple Weights for SUM. Notice that in the definition above, we assume that the input weights are assigned to the domain values of the attributes. Alternatively, the input weights could be assigned to the relation tuples, a convention that has been used in past work on ranked enumeration [40]. Since there are several reasonable semantics for interpreting a tuple-weight ranking for CQs with projections and/or self-joins [43], we elect to present our results for the case of attribute weights. We note that our results directly extend to the case of tuple weights for full self-join-free CQs where the semantics are clear. On the one hand, attribute weights can easily be transformed to tuple weights in linear time such that the weights of the query answers remain the same. This works by assigning each variable to one of the atoms that it appears in and computing the weight of a tuple by aggregating the weights of the assigned attribute values. Therefore, our hardness results for SUM orders directly extend to the case of tuple weights. On the other hand, our positive results on direct access (Section 5), selection (Section 7.2), and their extension to the case of FDs (Section 8.1) rely on algorithms that innately operate on tuple weights; thus, we cover those cases, too.

Direct Access vs. Selection. We now define two problems that both directly access ordered query answers. Since our goal is to classify the combination of CQs and orders by their tractability, we let those two define the problem. Specifically, a problem is defined by a CQ Q and a family of orders Π. The reason that we use a family of orders in the problem definition is that for the case of SUM, we do not distinguish between different weight functions in our classification. For LEX, we always consider the family of orders to contain only one specific (partial) lexicographic order.

Definition 2.1 (Direct Access). Let Q be a CQ and Π a family of total orders. The problem of *direct access* by Π takes as an input a database I and an order $\preceq$ from Π and constructs a data structure (called the *preprocessing* phase) that then allows access to a query answer $q \in Q(I)$ at any index k of the (implicit) array of query answers sorted by $\preceq$.

The essence of direct access is that after the preprocessing phase, we need to be able to support multiple such accesses. Notably, the values of k that are going to be requested afterward are not known during preprocessing.

Definition 2.2 (Selection). Let Q be a CQ and Π a family of total orders. The problem of *selection* by Π takes as an input a database I, an order $\preceq$ from Π, and asks for the query answer $q \in Q(I)$ at index k of the (implicit) array of query answers sorted by $\preceq$.

The problem of *selection* [9, 18, 19] is a computationally easier task that requires only a single direct access, hence does not make a distinction between preprocessing and access phases. A special case of the problem is finding the median query answer.

For both problems, if the index k exceeds the total number of answers, then the returned answer is "out-of-bound."

2.3 Complexity Framework and Sorting

We measure asymptotic complexity in terms of the size of the database n, while the size of the query is considered a constant. If the time for preprocessing is $O(f(n))$ and the time for each access is $O(g(n))$, then we denote that as $\langle f(n), g(n)\rangle$, where f, g are functions from $\mathbb{N}$ to $\mathbb{R}$. Note that, by definition, the problem of selection asks for a $\langle 1, g(n)\rangle$ solution.

Our goal for both problems is to achieve efficient access in time significantly smaller than (the worst case) $|Q(I)|$. For direct access, we consider the problem tractable if $\langle n \log n, \log n\rangle$ is possible, and for selection $\langle 1, n \log n\rangle$.

The model of computation is the standard RAM model with uniform cost measure. In particular, it allows for linear time construction of lookup tables, which can be accessed in constant time. We would like to point out that some past works [4, 14] have assumed that in certain variants of the model, sorting can be done in linear time [27]. Since we consider problems related to summation and sorting [20] where a linear-time sort would improve otherwise optimal bounds, we adopt a more standard assumption that sorting is comparison-based and possible only in quasilinear time. As a consequence, some upper bounds mentioned in this article are weaker than the original sources that assumed linear-time sorting [11, 14].

2.4 Hardness Hypotheses

All the lower bounds we prove are conditional on one or multiple of the following four hypotheses:

Hypothesis 1 (sparseBMM). *Two Boolean matrices A and B, represented as lists of their non-zero entries, cannot be multiplied in time $m^{1+o(1)}$, where m is the number of non-zero entries in A, B, and AB.*

A consequence of this hypothesis is that we cannot answer the query $Q(x, z) :- R(x, y), S(y, z)$ with quasilinear preprocessing and polylogarithmic delay. In more general terms, any self-join-free acyclic non-free-connex CQ cannot be enumerated with quasilinear[9] preprocessing time and polylogarithmic delay assuming the sparseBMM hypothesis [4, 7].

Hypothesis 2 (Hyperclique [1, 33]). *For every $k \geq 2$, there is no $O(m \operatorname{polylog} m)$ algorithm for deciding the existence of a $(k+1, k)$-hyperclique in a k-uniform hypergraph with m hyperedges, where a $(k+1, k)$-*hyperclique *is a set of $k+1$ vertices such that every subset of k elements is a hyperedge.*

When $k = 2$, this follows from the "δ-Triangle" hypothesis [1]. This is the hypothesis that we cannot detect a triangle in a graph in linear time [3]. When $k \geq 3$, this is a special case of the "(ℓ, k)-Hyperclique" hypothesis [33]. A known consequence is that Boolean cyclic and self-join-free CQs cannot be answered in quasilinear[9] time [11]. As a result, cyclic and self-join-free CQs do not admit enumeration with quasilinear preprocessing time and polylogarithmic delay assuming the Hyperclique hypothesis [11].

Hypothesis 3 (3sum [6, 38]). *Deciding whether there exist $a \in A, b \in B, c \in C$ from three sets of integers A, B, C, each of size $\Omega(m)$, such that $a + b + c = 0$ cannot be done in time $O(m^{2-\epsilon})$ for any $\epsilon > 0$.*

[9]Works in the literature [5, 7, 14] typically phrase this as linear, yet any logarithmic factor increase is still covered by the hypothesis.

In its simplest form, the 3SUM problem asks for three distinct real numbers a, b, c from a set S with m elements that satisfy $a + b + c = 0$. There is a simple $O(m^2)$ algorithm for the problem, but it is conjectured that, in general, no truly subquadratic solution exists [38]. The significance of this conjecture has been highlighted by many conditional lower bounds for problems in computational geometry [22] and within the class P in general [44]. Note that the problem remains hard even for integers provided that they are sufficiently large (i.e., in the order of $O(n^3)$) [38]. The hypothesis we use here has three different sets of numbers, but it is equivalent [6]. This lower bound has been confirmed in some restricted models of computation [2, 17].

HYPOTHESIS 4 (SETH [29]). *For the satisfiability problem with m variables and k variables per clause (k-SAT), if s_k is the infimum of the real numbers δ for which k-SAT admits an $O(2^{\delta m})$ algorithm, then* $\lim_{k\to\infty} s_k = 1$.

Intuitively, the *Strong Exponential Time Hypothesis* (SETH) states that the best possible algorithms for k-SAT approach $O(2^m)$ running time when k goes to infinity. SETH implies that the k-Dominating Set problem on a graph with m vertices cannot be solved in $O(m^{2-\epsilon})$ for $k \geq 3$ and any constant ϵ [39]. Based on that, it can be shown that counting the answers to a self-join-free acyclic CQ that is not free-connex cannot be done in $O(n^{2-\epsilon'})$ for any constant ϵ' [34].

2.5 Known Results for CQs

Eliminating Projection. We now provide some background that relates to the efficient handling of CQs. For a query with projections, a standard strategy is to reduce it to an equivalent one where techniques for acyclic full CQs can be leveraged. The following proposition, which is widely known and used [7], shows that this is possible for free-connex CQs.

PROPOSITION 2.3 (FOLKLORE). *Given a database instance I, a CQ Q, a join tree T of an inclusive extension of Q, and a subtree T' of T that contains all the free variables, we can compute in linear time a database instance I' over the schema of a CQ Q' that consists of the nodes of T' such that $Q(I) = Q'(I')$ and $|I'| \leq |I|$.*

This reduction is done by first creating a relation for every node in T using projections of existing relations, then performing the classic semi-join reduction by Yannakakis [46] to filter the relations of T' according to the relations of T, and then we can simply ignore all relations that do not appear in T' and obtain the same answers. Afterward, they can be handled efficiently, e.g., their answers can be enumerated with constant delay [4]. We refer the reader to recent tutorials [7, 16] for an intuitive illustration of the idea.

Ranked enumeration. Enumerating the answers to a CQ in ranked order is a special case of direct access where the accessed indexes are consecutive integers starting from 0. As it was recently shown [41], ranked enumeration for CQs is intimately connected to classic algorithms on finding the k-th shortest path in a graph. In contrast to direct access, ranked enumeration by SUM orders (which also includes lexicographic orderings as a special case) is possible with logarithmic delay after a linear-time preprocessing phase for all free-connex CQs [40]. In contrast, as we will show, that is not the case for direct access. Existing ranked-enumeration algorithms rely on priority queue structures that compare a minimal number of candidate answers to produce the ranked answers one-by-one in order. There is no straightforward way to extend them to the task of direct access where we may skip over a large number of answers to get to an arbitrary index k.

Direct Access. Carmeli et al. [14] devise a direct access structure (called "random access") and use it to uniformly sample CQ answers (called "random-order enumeration"). While it leverages the idea of using count statistics on the input tuples to navigate the space of query answers that had also been used in prior work on sampling [47], it decouples it from the random order requirement and advances it into direct access. The separation into a direct access component and a random

permutation (of indices) generated externally also allows sampling without replacement, which was not possible before. This direct access algorithm is also a significant simplification over a prior one by Brault-Baron [11]. We emphasize that even though these algorithms do not explicitly discuss the order of the answers, a closer look shows that they internally use and produce *some lexicographic order*.

Theorem 2.4 ([11, 14]). *Let Q be a CQ. If Q is free-connex, then direct access (in some order) is possible in $\langle n \log n, \log n \rangle$. Otherwise, if it is also self-join-free, then direct access (in any order) is not possible in $\langle n \text{ polylog } n, \text{polylog } n \rangle$, assuming sparseBMM and Hyperclique.*

Recent work by Keppeler [31] suggests another direct-access solution by lexicographic order, which also supports efficient insertion and deletion of input tuples. Given these additional requirements, the supported CQs are more limited and are only a subset of free-connex CQs called *q-hierarchical* [8]. This is a subclass of the well-known *hierarchical* queries with an additional restriction on the existential variables. As an example, the following CQs are not q-hierarchical even though they are free-connex: $Q_1(x, y) :- R_1(x), R_2(x, y), R_3(y)$ and $Q_2(x) :- R_1(x, y), R_2(y)$. For these queries, direct access is not supported by the solution of Keppeler [31], even though it is possible without the update requirements (as we show in Section 3).

All previous direct-access solutions of which we are aware have two gaps compared to this work: (1) they do not discuss which lexicographic orders (given by orderings of the free variables) are supported; (2) they do not support all possible lexicographic orders. We conclude this section with a short survey of existing solutions and their supported orders.

All prior direct-access solutions use some component that depends on the query structure and constrains the supported orders. The algorithm of Carmeli et al. [14, Algorithm 3] assumes that a join tree is given with the CQ, and the lexicographic order is *imposed by the join tree*. Specifically, it is an ordering of the variables achieved by a preorder depth-first traversal of the tree. As a result, it does not support any order that requires jumping back-and-forth between different branches of the tree. In particular, it does not support $Q_3(v_1, v_2, v_3, v_4) :- R(v_1, v_3), S(v_2, v_4)$ with the lexicographic order given by the increasing variable indices (we adopt this convention for all the examples below). We show how to handle this CQ and order in detail in Example **??**. The algorithm of Brault-Baron [11, Algorithm 4.3] assumes that an *elimination order* is given along with the CQ. The resulting lexicographic order is affected by that elimination order but is not exactly the same. This solution suffers from similar restrictions, and it does not support Q_3 either. The algorithm of Keppeler [31] assumes that a *q-tree* is given with the CQ, and the possible lexicographic orders are affected by this tree. Unlike the two earlier mentioned approaches, this algorithm can interleave variables from different atoms, yet cannot support some orders that are possible for the previous algorithms. As an example, it does not support $Q_4(v_1, v_2, v_3) :- R_1(v_1, v_2), R_2(v_2, v_3)$ as v_2 is highest in the hierarchy (the atoms containing it strictly subsume the atoms containing any other variable) and so it is necessarily the first variable in the q-tree and in the ordering produced.

Finally, we should mention that there exist queries and orders that require *both* jumping back-and-forth in the join tree *and* visiting the variables in an order different than any hierarchy. As a result, these are not supported by any previous solution. Two such examples are $Q_5(v_1, v_2, v_3, v_4, v_5) :- R_1(v_1, v_3), R_2(v_3, v_4), R_3(v_2, v_5)$ and $Q_6(v_1, v_2, v_3, v_4, v_5) :- R_1(v_1, v_2, v_4), R_2(v_2, v_3, v_5)$. In Section 3, we provide an algorithm that supports both of these CQs.

3 DIRECT ACCESS BY LEXICOGRAPHIC ORDERS

In this section, we answer the following question: For which underlying lexicographic orders can we achieve "tractable" direct access to ranked CQ answers, i.e., with quasilinear preprocessing and polylogarithmic time per answer?

Example 3.1 (No Direct Access). Consider the lexicographic order $L = \langle v_1, v_2, v_3 \rangle$ for the query $Q(v_1, v_2, v_3) :- R(v_1, v_3), S(v_3, v_2)$. Direct access to the query answers according to that order would allow us to "jump over" the v_3 values via binary search and essentially enumerate the answers to $Q'(v_1, v_2) :- R(v_1, v_3), S(v_3, v_2)$. However, we know that Q' is not free-connex and that is impossible to achieve enumeration with quasilinear preprocessing and polylogarithmic delay (if sparseBMM holds). Therefore, the bounds we are hoping for are out of reach for the given query and order. The core difficulty is that the joining variable v_3 appears *after* the other two in the lexicographic order.

We formalize this notion of "variable in the middle" to detect similar situations in more complex queries.

Definition 3.2 (Disruptive Trio). Let Q be a CQ and L a lexicographic order of its free variables. We say that three free variables v_1, v_2, v_3 are a *disruptive trio* in Q with respect to L if v_1 and v_2 are not neighbors (i.e., they do not appear together in an atom), v_3 is a neighbor of both v_1 and v_2, and v_3 appears after v_1 and v_2 in L.

As it turns out, for free-connex and self-join-free CQs, the tractable CQs are precisely captured by this simple criterion. Regarding self-join-free CQs that are not free-connex, their known intractability of enumeration implies that direct access is also intractable. This leads to the following dichotomy:

Theorem 3.3 (Direct Access by LEX). *Let Q be a CQ and L be a lexicographic order.*

- *If Q is free-connex and does not have a disruptive trio with respect to L, then direct access by L is possible in $\langle n \log n, \log n \rangle$.*
- *Otherwise, if Q is also self-join-free, then direct access by L is not possible in $\langle n \operatorname{polylog} n, \operatorname{polylog} n \rangle$ assuming sparseBMM and Hyperclique.*

Remark 1. Assume we are given a full CQ, and the lexicographic order we want to achieve is $\langle v_1, \ldots, v_m \rangle$. It was shown (in the context of ranked enumeration by lexicographic orders) that the absence of disruptive trios is equivalent to the existence of a reverse (α-)elimination order of the variables [11, Theorem 15]. That is, we need there to exist an atom that contains v_m and all of its neighbors (variables that share an atom with v_m), and if we remove v_m from the query, $v_1, \ldots, v_{m-1}$ should recursively be a reverse elimination order. For the base case, when $m = 1$, v_1 constitutes a reverse-elimination order.

Remark 2. On the positive side of Theorem 3.3, the preprocessing time is dominated by sorting the input relations, which we assume requires $O(n \log n)$ time. If we assume instead that sorting takes linear time (as assumed in some related work [11, 14, 27]), then the time required for preprocessing is only $O(n)$ instead of $O(n \log n)$.

In Section 3.1, we provide an algorithm for this problem for full acyclic CQs that have a particular join tree that we call *layered*. Then, we show how to find such a layered join tree whenever there is no disruptive trio in Section 3.2. In Section 3.3, we explain how to adapt our solution for CQs with projections, and in Section 3.4, we prove a lower bound, which establishes that our algorithm applies to *all* cases where direct access is tractable.

3.1 Layer-based Algorithm

Before we explain the algorithm, we first define one of its main components. A *layered join tree* is a join tree where each node belongs to a layer. The layer number is the last position of any of its

variables in the lexicographic order. Intuitively, "peeling" off the outermost (largest) layers must result in a valid join tree (for a hypergraph with fewer variables). To find such a join tree for a CQ Q, we may have to introduce hyperedges that are contained in those of $\mathcal{H}(Q)$ (this corresponds to taking the projection of a relation) or remove hyperedges of $\mathcal{H}(Q)$ that are contained in others (this corresponds to filtering relations that contain a superset of the variables). Thus, we define the layered join tree with respect to a hypergraph that is *inclusion-equivalent* (recall the definition of an inclusion-equivalent hypergraph from Section 2.1).

Definition 3.4 (Layered Join Tree). Let Q be a full acyclic CQ, and let $L = \langle v_1, \ldots, v_f \rangle$ be a lexicographic order. A *layered join tree* for Q with respect to L is a join tree of a hypergraph that is inclusion-equivalent to $\mathcal{H}(Q)$ where (1) every node V of the tree is assigned to layer $\max\{i \mid v_i \in V\}$, (2) there is exactly one node for each layer, and (3) for all $j \leq f$ the induced subgraph with only the nodes that belong to the first j layers is a tree.

Example 3.5. Consider the CQ $Q_3(v_1, v_2, v_3, v_4) :- R(v_1, v_3), S(v_2, v_4)$ and the lexicographic order $\langle v_1, v_2, v_3, v_4 \rangle$. To support that order, we first find an inclusion-equivalent hypergraph, shown in Figure 3(a). Notice that we added two hyperedges that are strictly contained in the existing ones and obtained a hypergraph corresponding to $R(v_1, v_3), R'(v_1), S(v_2, v_4), S'(v_2)$. A layered join tree constructed from that hypergraph is depicted in Figure 3(b). There are four layers, one for each node of the join tree. The layer of the node containing $\{v_1, v_3\}$ is 3, because v_3 appears after v_1 in the order and it is the third variable. If we remove the last layer, then we obtain a layered join tree for the induced hypergraph where the last variable v_4 is removed.

We now describe an algorithm that takes as an input a CQ Q, a lexicographic order L, and a corresponding layered join tree and provides direct access to the query answers after a preprocessing phase. For preprocessing, we leverage a construction from Carmeli et al. [14, Algorithm 2] and apply it to our layered join tree. For completeness, we briefly explain how it works below. Subsequently, we describe the access phase that takes into account the layers of the tree to accommodate the provided lexicographic order. We emphasize that the way we access the structure is different than that of the past work [14] and that this allows support of lexicographic orders that were impossible for the previous access routine (e.g., the order in Example 3.5).

Preprocessing. The preprocessing phase (1) creates a relation for every node of the tree, (2) removes dangling tuples, (3) sorts the relations, (4) partitions the relations into buckets, and (5) uses dynamic programming on the tree to compute and store certain counts.[10] After preprocessing, we are guaranteed that for all i, the node of layer i has a corresponding relation where each tuple participates in at least one query answer; this relation is partitioned into buckets by the assignment of the variables preceding i. In each bucket, we sort the tuples lexicographically by v_i. Each tuple is given a weight that indicates the number of different answers this tuple agrees with when only joining its subtree. The weight of each bucket is the sum of its tuple weights. We denote both by the function `weight`. Moreover, for every tuple t, we compute the sum of weights of the preceding tuples in the bucket, denoted by $\texttt{start}(t)$. We use $\texttt{end}(t)$ for the sum that corresponds to the tuple following t in the same bucket; if t is last, then we set this to be the bucket weight. If we think of the query answers in the subtree sorted in the order of v_i values, then `start` and `end` distribute the indices between 0 and the bucket weight to tuples. The number of indices within the range of each tuple corresponds to its weight.

[10] The same count statistics are also leveraged in Reference [47, Section 4.2] in the context of sampling.

R'	w	s	e
a_1	8	0	8
a_2	8	8	16

S'	w	s	e
b_1	3	0	3
b_2	1	3	4

R		w	s	e
a_1	c_1	1	0	1
a_1	c_2	1	1	2
a_2	c_2	1	0	1
a_2	c_3	1	1	2

S		w	s	e
b_1	d_1	1	0	1
b_1	d_2	1	1	2
b_1	d_3	1	2	3
b_2	d_4	1	0	1

Fig. 4. Example 3.6: The result of the preprocessing phase on Q_3, the layered join tree (Figure 3(b)) and an example database. The weight, start index, and end index for each tuple are abbreviated in the figure as w, s, and e, respectively.

Example 3.6 (Continued). The result of the preprocessing phase on an example database for our query Q_3 is shown in Figure 4. Notice that R has been split into two buckets according to the values of its parent R', one for value a_1 and one for a_2. For tuple $(a_1) \in R'$, we have $\texttt{weight}((a_1)) = 8$, because this is the number of answers that agree on that value in its subtree: The left subtree has two such answers that can be combined with any of the four possible answers of the right subtree. The start index of tuple $(b_1, d_3) \in S$ is the sum of the previous weights within the bucket: $\texttt{start}((b_1, d_3)) = \texttt{weight}((b_1, d_1)) + \texttt{weight}((b_1, d_2)) = 1 + 1 = 2$. Not shown in the figure is that every bucket stores the sum of weights it contains.

Access. The access phase works by going through the tree layer-by-layer. When resolving a layer i, we select a tuple from its corresponding relation, which sets a value for the i-th variable in L and also determines a bucket for each child. Then, we conceptually erase the node of layer i and its outgoing edges.

The access algorithm maintains a directed forest and an assignment to a prefix of the variables. Each tree in the forest represents the answers obtained by joining its relations. Each root contains a single bucket that agrees with the already assigned values, thus every answer agrees on the prefix. Due to the running intersection property, different trees cannot share unassigned variables. As a consequence, any combination of answers from different trees can be added to the prefix assignment to form an answer to Q. The answers obtained this way are exactly the answers to Q that agree with the already set assignment. Since we start with a layered join tree, we are guaranteed that, at each step, the next layer (which corresponds to the variable following the prefix for which we have an assignment) appears as a root in the forest.

Recall that from the preprocessing phase, the weight of each root is the number of answers in its tree. When we are at layer i, we have to take into account the weights of all the other roots to compute the number of query answers for a particular tuple. More specifically, the number of answers to Q containing the already selected attributes (smaller than i) and some v_i value contained in a tuple is found by multiplying the tuple weight with the weights of all other roots. That is because the answers from all trees can be combined into a query answer. Let t be the selected tuple when resolving the i-th layer. The number of answers to Q that have a value of $L[i]$ smaller than that of t and a value of $L[j]$ equal to that of t for all $j < i$ is then:

$$\sum_{t'} \left(\texttt{weight}(t') \prod_{r \in \text{roots}} \texttt{weight}(r) \right),$$

where t' ranges over tuples preceding t in its bucket. Denote by factor the product of all root weights. Then, we can rewrite as:

$$\left(\sum_{t'} \texttt{weight}(t') \right) \left(\prod_{r \in \text{roots}} \texttt{weight}(r) \right) = \texttt{start}(t) \cdot \text{factor}\,.$$

Therefore, when resolving layer i, we select the last tuple t such that the index we want to access is at least $\texttt{start}(t) \cdot \text{factor}$.

ALGORITHM 1: Lexicographic Direct-Access

```
1  if k ≥ weight(root) then
2      return "out-of-bound"
3  bucket[1] = root
4  factor = weight(root)
5  for i=1,...,f do
6      factor = factor/weight(bucket[i])
7      pick t ∈ bucket[i] s.t. start(t) · factor ≤ k < end(t) · factor
8      k = k − start(t) · factor
9      for child V of layer i do
10         get the bucket b ∈ V agreeing with the selected tuples
11         bucket[layer(V)] = b
12         factor = factor · weight(b)
13 return the answer agreeing with the selected tuples
```

Algorithm 1 summarizes the process we described where k is the index to be accessed and f is the number of variables. Iteration i resolves layer i. Pointers to the selected buckets from the roots are kept in a bucket array. The product of the weights of all roots is kept in a factor variable. In each iteration, the variable k is updated to the index that should be accessed among the answers that agree with the already selected attribute values. Note that bucket$[i]$ is always initialized when accessed, since layer i is guaranteed to be a child of a smaller layer.

Example 3.7 (Continued). We demonstrate how the access algorithm works for index $k = 12$. When resolving R', the tuple (a_2) is chosen, since $8 \cdot 1 \leq 12 < 16 \cdot 1$; then, the single bucket in S' and the bucket containing a_2 in R are selected. The next iteration resolves S'. When it reaches line 7, $k = 12 - 8 = 4$ and factor $= 2$. As $0 \cdot 2 \leq 4 < 3 \cdot 2$, the tuple (b_1) is selected. Next, R is resolved, which we depict in Figure 5. The current index is $k = 4 - 0 = 4$. The weights of the other roots (only S here) gives us factor $= 3$. To make our choice in R, we multiply the weights of the tuples by factor $= 3$. Then, we find that the index k we are looking for falls into the range of (a_2, c_3), because $1 \cdot 3 \leq 4 < 2 \cdot 3$. Next, S is resolved, $k = 4 - 1 \cdot 3 = 1$, and factor $= 1$. As $1 \cdot 1 \leq 1 < 2 \cdot 1$, the tuple (b_1, d_2) is selected. Overall, answer number 12 (the 13th answer) is (a_2, b_1, c_3, d_2).

Lemma 3.8. *Let Q be a full acyclic CQ and $L = \langle v_1, \ldots, v_f \rangle$ be a lexicographic order. If there is a layered join tree for Q with respect to L, then direct access is possible in $\langle n \log n, \log n \rangle$.*

Proof. The correctness of Algorithm 1 follows from the discussion above. For the time complexity, note that it contains a constant number of operations (assuming the number of attributes f is fixed). Line 7 can be done in logarithmic time using binary search, while all other operations only require constant time in the RAM model. Thus, we obtain direct access in logarithmic time per answer after the quasilinear preprocessing (dominated by sorting). □

Remark 3 (Inverted Access). A straightforward adaptation of Algorithm 1 can be used to achieve *inverted access*: Given a query result as the input, we return its index according to the lexicographic order. Algorithm 2 is almost the same as Algorithm 1 except that the choices in each iteration are made according to the given answer and the corresponding index is constructed (instead of the

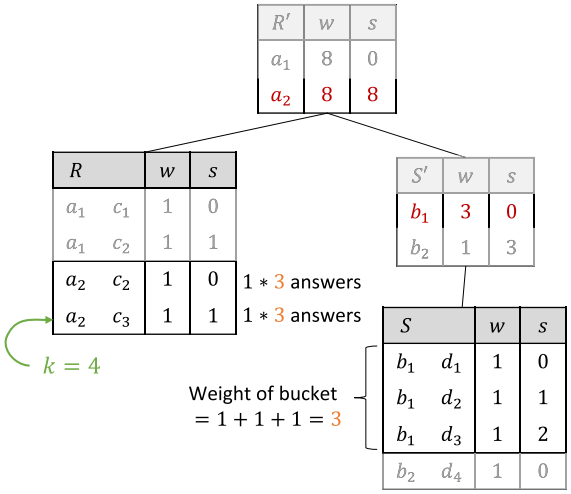


Fig. 5. Example 3.7: Illustration of an iteration of the access phase where layer 3 corresponding to R is resolved.

opposite). The algorithm runs in constant time per answer, since every operation can be done within that time (unlike Algorithm 1, there is no need for binary search here).

Another adaptation of Algorithm 2 can give us a form of inverted access for the cases when the given answer does not exist. That is, instead of returning "not-an-answer," we want to return the next answer in the lexicographic order. The first time a tuple t is not found in Line 7 of Algorithm 2, we select the first tuple in the bucket that is larger than t, and in all following iterations, we always select the first tuple in the bucket. If there is no tuple larger than t, then we revert the previous iteration and select the next tuple there (compared to what we selected before). If no such tuple exists, then we again revert the previous iteration and so on. If there are no previous iterations, then we were asked to access a tuple larger than the last answer, so we return an appropriate message. The algorithm described here takes logarithmic time, as we can use binary search to find the tuple following our target tuple in each bucket.

3.2 Finding Layered Join Trees

We now have an algorithm that can be applied whenever we have a layered join tree. We next show that the existence of such a join tree relies on the disruptive trio condition we introduced earlier. In particular, if no disruptive trio exists, then we are able to construct a layered join tree for full acyclic CQs.

Lemma 3.9. *Let Q be a full acyclic CQ and L be a lexicographic order. If Q does not have a disruptive trio with respect to L, then there is a layered join tree for Q with respect to L.*

Proof. We show by induction on i that there exists a layered join tree for the hypergraph containing the hyperedges $\{V \cap \{v_1, \ldots, v_i\} \mid V \in \texttt{atoms}(Q)\}$ with respect to the prefix of L containing its first i elements. The induction base is the tree that contains the node $\{v_1\}$ and no edges.

In the inductive step, we assume a layered join tree with $i-1$ layers for $\{V \cap \{v_1, \ldots, v_{i-1}\} \mid V \in \texttt{atoms}(Q)\}$, and we build a layer on top of it. Denote by $\mathcal{V}$ the sets of $\{V \cap \{v_1, \ldots, v_i\} \mid V \in \texttt{atoms}(Q)\}$ that contain v_i (these are the sets that need to be included in the new layer). First note that $\mathcal{V}$ is acyclic. Indeed, by the running intersection property, the join tree for $\mathcal{H}(Q)$ has a

ALGORITHM 2: Lexicographic Inverted-access

```
1  k = 0
2  bucket[1] = root
3  factor = weight(root)
4  for i=1,...,f do
5  |  factor = factor/weight(bucket[i])
6  |  select t ∈ bucket[i] agreeing with the answer
7  |  if no such t exists then
8  |  |  return "not-an-answer"
9  |  k = k + start(t) · factor
10 |  for child V of layer i do
11 |  |  get the bucket b ∈ V agreeing with the answer
12 |  |  bucket[layer(V)] = b
13 |  |  factor = factor · weight(b)
14 return k
```

subtree with all the nodes that contain v_i. By taking this subtree and projecting out all variables that occur after v_i in L, we get a join tree for an inclusion-equivalent hypergraph to $\mathcal{V}$, and its existence proves that $\mathcal{V}$ is acyclic.

We next claim that some set in $\mathcal{V}$ contains all the others; that is, there exists $V_m \in \mathcal{V}$ such that for all $V \in \mathcal{V}$, we have that $V \subseteq V_m$. Consider a join tree for $\mathcal{V}$. Every variable $v \in \mathcal{V}$ defines a subtree T_v induced by the nodes that contain this variable. If two variables x, y are neighbors, then their subtrees T_x, T_y share a node. It is known that every collection of subtrees of a tree satisfies the *Helly property* [25]: If every two subtrees share a node, then some node is shared by all subtrees. In particular, since $\mathcal{V}$ is acyclic, if every two variables of $\mathcal{V}$ are neighbors, then some element of $\mathcal{V}$ contains all variables that appear in (elements of) $\mathcal{V}$. Thus, if, by way of contradiction, there is no such V_m, then there exist two non-neighboring variables v_a and v_b that appear in (elements of) $\mathcal{V}$. Since v_i appears in all elements of $\mathcal{V}$, this means that there exist $V_a, V_b \in \mathcal{V}$ with $\{v_a, v_i\} \subseteq V_a$ and $\{v_b, v_i\} \subseteq V_b$. Since v_a and v_b are not neighbors, these three variables are a disruptive trio with respect to L: v_a and v_b are both neighbors of the later variable v_i. The existence of a disruptive trio contradicts the assumption of the lemma we are proving, and so we conclude that there is $V_m \in \mathcal{V}$ such that for all $V \in \mathcal{V}$, we have that $V \subseteq V_m$.

With V_m at hand, we can now add the additional layer to the tree given by the inductive hypothesis. By the inductive hypothesis, the layered join tree with $i-1$ layers contains the hyperedge $V_m \cap \{v_1, \ldots, v_{i-1}\} = V_m \setminus \{v_i\}$. We insert V_m with an edge to the node containing $V_m \setminus \{v_i\}$.

This results in the join tree we need: (1) the hyperedges $\{V \cap \{v_1, \ldots, v_i\} \mid V \in \texttt{atoms}(Q)\}$ are all contained in nodes, since the ones that do not appear in the tree from the inductive hypothesis are contained in the new node; (2) it is a tree, since we add one leaf to an existing tree; and (3) the running intersection property holds, since the added node is connected to all of its variables that already appear in the tree. □

Lemmas 3.8 and 3.9 give a direct-access algorithm for full acyclic CQs and lexicographic orders without disruptive trios.

3.3 Supporting Projection

Next, we show how to support CQs that have projections. A free-connex CQ can be efficiently reduced to a full acyclic CQ using Proposition 2.3. We next show that the resulting CQ contains no disruptive trio if the original CQ does not.

Lemma 3.10. *Given a database instance I, a free-connex CQ Q, and a lexicographic order L with no disruptive trio with respect to L, we can compute in linear time a database instance I' and a full acyclic CQ Q' with no disruptive trio with respect to L such that $Q'(I') = Q(I)$, $|I'| \leq |I|$, and Q' does not depend on I or I'.*

Proof. Let Q be a free-connex CQ, and let T be an ext-$\texttt{free}(Q)$-connex tree for Q where T' is the subtree of T that contains exactly the free variables.

First, we claim that two free variables are neighbors in T iff they are neighbors in T'. The "if" direction is immediate, since T' is contained in T. We show the other direction. Let u and v be free variables of Q that are neighbors in T. That is, there is a node V_T in T that contains them both. Consider the unique path from V to any node in T' such that only the last node on the path, which we denote $V_{T'}$, is in T'. Since both variables appear in T' and in V, by the running intersection property, both variables appear in $V_{T'}$. Thus, u and v are also neighbors in T'.

Since the definition of disruptive trios depends only on neighboring pairs of free variables, an immediate consequence of the claim from the previous paragraph is that there is a disruptive trio in T iff there is a disruptive trio in T'. Next, we can simply use Proposition 2.3 to reduce Q to the full acyclic CQ where the atoms are exactly the nodes of T'. □

By combining Lemmas 3.8 to 3.10, we conclude an efficient algorithm for free-connex CQs and orders with no disruptive trios. The next lemma summarizes our results so far.

Lemma 3.11. *Let Q be a free-connex CQ and L be a lexicographic order. If Q does not have a disruptive trio with respect to L, then direct access by L is possible in $\langle n \log n, \log n \rangle$.*

3.4 Lower Bound for Conjunctive Queries

Next, we show that our algorithm supports all tractable cases (for self-join-free CQs); we prove that all unsupported cases are intractable. We base our hardness results on the known hardness of enumeration for non-free-connex CQs [5, 11] through a reduction that uses direct access to enumerate the answers projected on a prefix of the variables. Adapting our notation to enumeration, we say that an enumeration problem is in $\langle p(n), f(n) \rangle$ if there is an algorithm that solves this problem with $p(n)$ time before the first answer and $f(n)$ time between consecutive answers.

Lemma 3.12. *Let Q be a self-join-free CQ, L be a lexicographic order, and Q' be the same as Q but with free variables L' for some prefix L' of L. If direct access for Q by L is possible in $\langle p(n), f(n) \rangle$ for some functions p, f, then enumeration of the answers to Q' is possible in $\langle p(n), f(n) \log n \rangle$.*

Proof. We show how to enumerate the unique assignments of the free variables of Q' given the direct access algorithm for Q. First, we perform the preprocessing step in $O(p(n))$. Then, we perform the following starting with $i = 0$ and until there are no more answers. We access the answer at index i and print its assignment to the variables L'. Then, we set i to be the index of the next answer that assigns different values to L' and repeat. Finding the next index can be done with a logarithmic number of direct access calls using binary search. □

We now exploit that, for CQs with disruptive trios, we can always find a prefix that is not connex. Therefore, enumerating the query answers projected on that prefix via direct access leads to the enumeration of a non-free-connex CQ, where existing lower bounds apply.

Lemma 3.13. *Let Q be a self-join-free acyclic CQ and L be a lexicographic order. If Q has a disruptive trio with respect to L, then direct access by L is not possible in $\langle n \text{ polylog } n, \text{polylog } n\rangle$, assuming sparseBMM.*[11]

Proof. Let v_1, v_2, v_3 be a disruptive trio in L. We take L' to be the prefix of L that ends in v_2. Then, v_1, v_3, v_2 is an L'-path, or in other words, the hypergraph of Q is not L'-connex. Now, we define a new CQ Q' so it has the same body as Q but its free variables are L'. Thus, Q' is acyclic but not free-connex. Assuming that direct access for Q is possible in $\langle n \text{ polylog } n, \text{polylog } n\rangle$, we use Lemma 3.12 to enumerate the answers of Q' in $\langle n \text{ polylog } n, \text{polylog } n\rangle$, which is known to contradict sparseBMM [5]. □

By combining Lemmas 3.11 and 3.13 together with the known hardness results for non-free-connex CQs (Theorem 2.4), we prove the dichotomy given in Theorem 3.3: Direct access by a lexicographic order for a self-join-free CQ is possible with quasilinear preprocessing and polylogarithmic time per answer if and only if the query is free-connex and does not have a disruptive trio with respect to the required order.

4 DIRECT ACCESS BY PARTIAL LEXICOGRAPHIC ORDERS

We now investigate the case where the desired lexicographic order is *partial*, i.e., it contains only some of the free variables. This means that there is no particular order requirement for the rest of the variables. One way to achieve direct access to a partial order is to complete it into a full lexicographic order and then leverage the results of the previous section. If such completion is impossible, then we have to consider cases where tie-breaking between the non-ordered variables is done in an arbitrary way. However, we will show in this section that the tractable partial orders are precisely those that can be completed into a full lexicographic order. In particular, we will prove the following dichotomy that also gives an easy-to-detect criterion for the tractability of direct access.

Theorem 4.1. *Let Q be a CQ and L be a partial lexicographic order.*

- *If Q is free-connex and L-connex and does not have a disruptive trio with respect to L, then direct access by L is possible in $\langle n \log n, \log n\rangle$.*
- *Otherwise, if Q is also self-join-free, then direct access by L is not possible in $\langle n \text{ polylog } n, \text{polylog } n\rangle$, assuming sparseBMM and Hyperclique.*

Example 4.2. Consider the CQ $Q :- R(x, y), S(y, z)$. If the free variables are exactly x and z, then the query is not free-connex, and so it is intractable. Next assume that all variables are free. If $L = \langle x, z\rangle$, then the query is not L-connex, and so it is intractable. If $L = \langle x, z, y\rangle$, then x, z, y is a disruptive trio, thus the query is intractable. However, if $L = \langle x, y, z\rangle$ or $L = \langle z, y\rangle$, then the query is free-connex, L-connex, and has no disruptive trio, so it is tractable.

4.1 Tractable Cases

For the positive side, we can solve our problem efficiently if the CQ is free-connex and there is a completion of the lexicographic order to all free variables with no disruptive trio. Lemma 4.4

[11] In fact, this lemma holds also for cyclic CQs, as it can be shown that Boolean matrix multiplication can be encoded in any CQ that contains a free-path regardless of its acyclicity. However, this is not formally stated in previous work, and we prefer not to complicate the proof with the technical details of the reduction. We chose here to limit the statement to acyclic CQs, as cyclic CQs are already known to be hard if we assume Hyperclique. The direct reduction that applies also to cyclic CQs can be found in the conference version of this article [13].

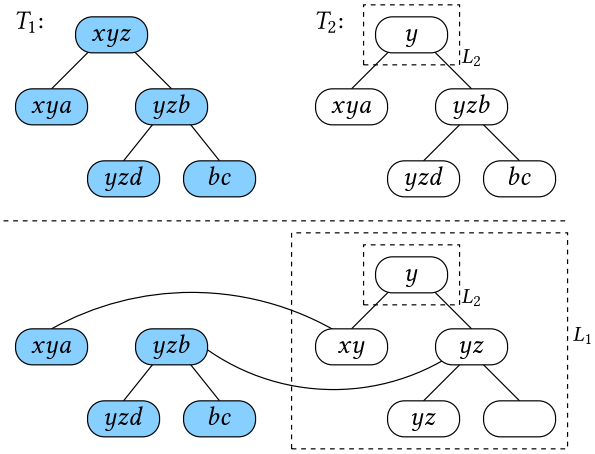


Fig. 6. Example for the construction from Proposition 4.3 for the CQ $Q(x, y, z) :- R_1(x, y, a), R_2(y, z, b), R_3(b, c), R_4(y, z, d)$ with $L_1 = \{x, y, z\}$ and $L_2 = \{y\}$.

identifies these cases with a connexity criterion. To prove it, we first need a way to combine two different connexity properties. The proof of the following proposition uses ideas from a proof of the characterization of free-connex CQs in terms of the acyclicity of the hypergraph obtained by including a hyperedge with the free variables [7].

Proposition 4.3. *If a CQ Q is both L_1-connex and L_2-connex where $L_2 \subseteq L_1$, then there exists a join tree T of an inclusive extension of Q with a subtree T_1 containing exactly the variables L_1 and a subtree T_2 of T_1 contains exactly the variables L_2.*

Proof. We describe a construction of the required tree. Figure 6 demonstrates our construction. We use two different characterizations of connexity. Since Q is L_2-connex, it has an ext-L_2-connex tree T_2. Since Q is L_1-connex, there is a join-tree T_1 for the atoms of Q and its head. Let $T_2[L_1]$ be T_2 where the variables that are not in L_1 are deleted from all nodes. That is, for every node $V \in T_2$, its variables are replaced with $\mathtt{var}(V) \cap L_1$. Denote by $\mathcal{V}$ all neighbors of the head in T_1, and denote by T_1^- the graph T_1 after the deletion of the head node. Taking both $T_2[L_1]$ and T_1^- and connecting every node $V_1 \in \mathcal{V}$ with a node V_2 of $T_2[L_1]$ such that $\mathtt{var}(V_1) \cap L_1 = \mathtt{var}(V_2)$ gives us the tree we want. Such a node exists in $T_2[L_1]$, since every node of T_1^- represents an atom of Q, and every atom of Q is contained in some node of T_2. The subtree $T_2[L_1]$ contains exactly V_1, and, since this subtree comes from an ext-L_2-connex tree, it has a subtree containing exactly L_1. It is easy to verify that the result is a tree, and we can show that the running intersection property holds in the united graph, since it holds for T_1 and T_2. □

We are now in a position to show the following:

Lemma 4.4. *Let Q be a CQ and L be a partial lexicographic order. If Q is free-connex and L-connex and does not have a disruptive trio with respect to L, then there is an ordering L^+ of $\mathtt{free}(Q)$ that starts with L such that Q has no disruptive trio with respect to L^+.*

Proof. According to Proposition 4.3, there is a join tree T (of an inclusive extension of Q) with a subtree $T_{\mathtt{free}}$ containing exactly the free variables, and a subtree T_L of $T_{\mathtt{free}}$ containing exactly the L variables. We assume that T_L contains at least one node; otherwise (this can only happen in case L is empty), we can introduce a node with no variables to all of T, $T_{\mathtt{free}}$, and T_L and connect it to any one node of $T_{\mathtt{free}}$. We describe a process of extending L while traversing $T_{\mathtt{free}}$. Consider the

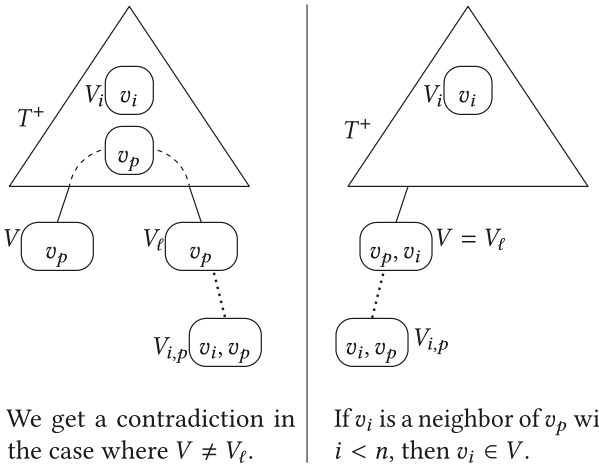


Fig. 7. The induction step in Lemma 4.4

nodes of T_L as handled, and initialize $L^+ = L$. Then, repeatedly handle a neighbor of a handled node until all nodes are handled. When handling a node, append to L^+ all of its free variables that are not already there. Since the join tree is connected and includes all query variables, L^+ will necessarily contain $\texttt{free}(Q)$ at the end of the process. We prove by induction that Q has no disruptive trio w.r.t. any prefix of L^+. The base case is guaranteed by the premises of this lemma, since L (hence all of its prefixes) has no disruptive trio.

Let v_p be a new variable added to a prefix $v_1, \ldots, v_{p-1}$ of L^+. Let T^+ be the subtree of $T_{\texttt{free}}$ with the handled nodes when adding v_p to L^+, and let $V \notin T^+$ be the node being handled. Note that, since v_p is being added, $v_p \in V$ but v_p is not in any node of T^+.

We first claim that every neighbor v_i of v_p with $i < p$ is in V. Our arguments are illustrated in Figure 7. Since v_i and v_p are neighbors, they appear together in a node $V_{i,p}$ outside of T^+. Let V_i be a node in T^+ containing v_i (such a node exists, since v_i appears before v_p in L^+). Consider the path from $V_{i,p}$ to V_i. Let V_ℓ be the last node of this path not in T^+. If $V_\ell \neq V$, then the path between V_ℓ and V goes only through nodes of T^+ (except for the end-points). Thus, concatenating the path from $V_{i,p}$ to V_ℓ with the path from V_ℓ to V results in a simple path. By the running intersection property, all nodes on this path contain v_p. In particular, the node following V_ℓ contains v_p in contradiction to the fact that v_p does not appear in T^+. Therefore, $V_\ell = V$. By the running intersection property, since V is on the path between V_i and $V_{i,p}$, we have that V contains v_i.

We now prove the induction step. We know by the inductive hypothesis that $v_1, \ldots, v_{p-1}$ have no disruptive trio. Assume by way of contradiction that appending v_p introduces a disruptive trio. Then, there are two variables v_i, v_j with $i < j < p$ such that v_i, v_p are neighbors, v_j, v_p are neighbors, but v_i, v_j are not neighbors. As we proved, since v_i and v_j are neighbors of v_p preceding it, we have that all three of them appear in the handled node V. This is a contradiction to the fact that v_i and v_j are not neighbors. □

The positive side of Theorem 4.1 is obtained by combining Lemma 4.4 with Theorem 3.3.

4.2 Intractable Cases

For the negative part, we prove a generalization of Lemma 3.13. Recall that according to Lemma 3.12, we can use lexicographic direct access to enumerate the answers to a CQ with a prefix of the ordered free variables. Similarly to Section 3.4, our goal is to find a "bad" prefix that

does not allow efficient enumeration. For non-L-connex CQs, this is easy, since L itself is such a prefix.

Lemma 4.5. *Let Q be an acyclic self-join free CQ and L be a partial lexicographic order. If Q has a disruptive trio or Q is not L-connex, then there exists a self-join-free acyclic non-free-connex CQ Q' such that: If direct access for Q is possible in $\langle n \operatorname{polylog} n, \operatorname{polylog} n\rangle$, then enumeration for Q' is possible in $\langle n \operatorname{polylog} n, \operatorname{polylog} n\rangle$.*

Proof. If Q is not L-connex, then we use Lemma 3.12 with $L' = L$. If L has a disruptive trio v_1, v_2, v_3, then we take L' to be the prefix of L that ends in v_2. Then, v_1, v_3, v_2 is an L'-path, meaning that the body of Q is not L'-connex. Thus, we can use Lemma 3.12 in that case, too. □

It is known that, assuming sparseBMM, self-join-free non-free-connex CQs cannot be answered with polylogarithmic time per answer after quasilinear preprocessing time. Thus, we conclude from Lemma 4.5 that self-join-free acyclic CQs with disruptive trios or that are not L-connex do not have partial lexicographic direct access within these time bounds either. The case that Q is cyclic is hard, since even finding any answer for cyclic CQs is not possible efficiently assuming Hyperclique.

5 DIRECT ACCESS BY SUM OF WEIGHTS

We now consider direct access for the more general orderings based on SUM (the *sum* of free-variable weights). As with lexicographic orderings, we are able to exhaustively classify tractability for the self-join-free CQs, even those with projections. We will show that direct access for SUM is significantly harder and tractable only for a small class of queries.

5.1 Overview of Results

The main result of this section is a dichotomy for direct access by SUM orders:

Theorem 5.1 (Direct Access by SUM). *Let Q be a CQ.*

- *If Q is acyclic and an atom of Q contains all the free variables, then direct access by SUM is possible in $\langle n \log n, 1\rangle$.*
- *Otherwise, if Q is also self-join-free, then direct access by SUM is not possible in $\langle n \operatorname{polylog} n, \operatorname{polylog} n\rangle$, assuming 3sum and Hyperclique.*

For the positive part of the above theorem, we will see that we are able to materialize the query answers and keep them in a sorted array that supports direct access in constant time. The proof of the negative part requires the query answers to express certain combinations of weights. If the query contains *independent* free variables, then its answers may contain all possible combinations of their corresponding attribute weights. We will thus rely on this independence measure to identify hard cases.

Definition 5.2 (Independent Free Variables). A set of vertices $V_i \subseteq V$ of a hypergraph $\mathcal{H}(V, E)$ is called independent iff no pair of these vertices appears in the same hyperedge, i.e., $|V_i \cap e| \leq 1$ for all $e \in E$. For a CQ Q, we denote by $\alpha_{\text{free}}(Q)$ the maximum number of variables among $\texttt{free}(Q)$ that are independent in $\mathcal{H}(Q)$.

Intuitively, we can construct a database instance where each independent free variable is assigned to n different domain values with n different weights. By appropriately choosing the assignment of the other variables, all possible $n^{\alpha_{\text{free}}(Q)}$ combinations of these weights will appear in the query answers. Providing direct access then implies that we can retrieve these sums in

ranked order. We later use this to show that direct access on certain CQs allows us to solve 3SUM efficiently.

Example 5.3. For $Q(x, y, z) :- R(x, y), S(y, z), T(z, u)$, we have $\alpha_{\text{free}}(Q) = 2$, namely, for variables $\{x, z\}$. Let the binary relation R be $[1, n] \times \{0\}$, i.e., the cross product between the set of values from 1 to n with the single value 0. If we also set $S = \{0\} \times [1, n]$ and $T = [1, n] \times \{0\}$, then the query answers are the n^2 assignments of (x, y, z) to $[1, n] \times [1, n] \times \{0\}$. The n values of x and z can be, respectively, assigned to any real-valued weights such that direct access on Q retrieves their ith sum in ranked order.

Our independence measure $\alpha_{\text{free}}(Q)$ is related to the classification of Theorem 5.1 in the following way:

LEMMA 5.4. *For an acyclic CQ Q, an atom contains all the free variables iff $\alpha_{free}(Q) \leq 1$.*

PROOF. The "only if" part of $\alpha_{\text{free}}(Q) > 1$ follows immediately from Definition 5.2.

For $\alpha_{\text{free}}(Q) = 1$ and acyclic query Q, we prove that there is an atom $R_f(\mathbf{X}_f)$ that contains all the free variables. First note that for $|\texttt{free}(Q)| = 1$ this is trivially true. For $|\texttt{free}(Q)| > 1$, let V be a node in the join tree (corresponding to some atom of Q) that contains the maximum number of free variables, and assume for the sake of contradiction that there exists a free variable y with $y \notin V$. We use $\mathcal{V}_y$ to denote the set of nodes in the join tree that contain variable y; thus, $V \notin \mathcal{V}_y$. From Q being acyclic follows that the nodes in $\mathcal{V}_y$ form a connected graph and there exists a node V' that lies on every path from V to a node in $\mathcal{V}_y$. Since $\alpha_{\text{free}}(Q) = 1$, each variable $x \in V$ must appear together with y in some query atom, implying that x appears in some node $V'' \in \mathcal{V}_y$. From that and the running intersection property follows that x must also appear in V', since V' lies on the path from V to any such V''. Hence V' contains y and all the V variables, violating the maximality assumption for V.

For $\alpha_{\text{free}}(Q) = 0$, Q is a Boolean query and any atom trivially contains the empty set. □

Therefore, the dichotomy of Theorem 5.1 can equivalently be stated using $\alpha_{\text{free}}(Q) \leq 1$ as a criterion. We chose to use the other criterion (all free variables contained in one atom) in the statement of our theorem statement, as it is more straightforward to check. In the next section, we proceed to prove our theorem by showing intractability for all queries with $\alpha_{\text{free}}(Q) > 1$ and a straightforward algorithm for $\alpha_{\text{free}}(Q) \leq 1$.

5.2 Proofs

For the hardness results, we rely mainly on the 3SUM hypothesis. To more easily relate our direct-access problem to 3SUM, which asks for the existence of a particular sum of weights, it is useful to define an auxiliary problem:

Definition 5.5 (Weight Lookup). Given a CQ Q and a weight function w over its possible answers, *weight lookup* takes as an input a database I and $\lambda \in \mathbb{R}$ and returns the first index of a query answer $q \in Q(I)$ with $w(q) = \lambda$ in the array of answers sorted by w or "none" if no such answer exists.

The following lemma associates direct access with weight lookup via binary search on the query answers:

LEMMA 5.6. *For a CQ Q, if the k-th query answer ordered by a weight function w can be directly accessed in $O(g(n))$ time for every k, then weight lookup for Q and w can be performed in $O(g(n) \log n)$.*

PROOF. We use binary search on the sorted array of query answers. Each direct access returns a query answer whose weight can be computed in $O(1)$. Thus, in a logarithmic number of accesses,

we can find the first occurrence of the desired weight. Since the number of answers is polynomial in n, the number of accesses is $O(\log n)$ and each one takes $O(g(n))$ time. □

Lemma 5.6 implies that, whenever we are able to support efficient direct access on the sorted array of query answers, weight lookup increases time complexity only by a logarithmic factor, i.e., it is also efficient. The main idea behind our reductions is that via weight lookups on a CQ with an appropriately constructed database, we can decide the existence of a zero-sum triplet over three distinct sets of numbers, thus hardness follows from 3SUM. First, we consider the case of three independent variables that are free. These three variables are able to simulate a three-way Cartesian product in the query answers. This allows us to directly encode the 3SUM triplets using attribute weights, obtaining a lower bound for direct access.

LEMMA 5.7. *If a CQ Q is self-join-free and $\alpha_{free}(Q) \geq 3$, then direct access by SUM is not possible in $\langle n^{2-\epsilon}, n^{2-\epsilon}\rangle$ for any $\epsilon > 0$ assuming 3SUM.*

PROOF. Assume for the sake of contradiction that the lemma does not hold. We show that this would imply an $O(n^{2-\epsilon})$-time algorithm for 3SUM. To this end, consider an instance of 3SUM with integer sets A, B, and C of size n, given as arrays. We reduce 3SUM to direct access over the appropriate query and input instance by using a construction similar to Example 5.3. Let x, y, and z be free and independent variables of Q, which exist because $\alpha_{\text{free}}(Q) \geq 3$. We create a database instance where x, y, and z take on each value in $[1, n]$, while all the other attributes have value 0. This ensures that Q has exactly n^3 answers—one for each (x, y, z) combination in $[1, n]^3$, no matter the number of atoms and the variables they contain. To see this, note that, since x, y, and z are independent, no pair of them appears together in an atom. Also, since Q is self-join-free, each relation appears once in the query, hence contains at most one of x, y, and z. Thus, each relation either contains 1 tuple (if neither x, y, nor z is present) or n tuples (if one of x, y, or z is present). No matter on which attributes these relations are joined (including Cartesian products), the output result is always the "same" set $[1, n]^3 \times \{0\}^f$ of size n^3, where f is the number of free variables other than x, y, and z. (We use the term "same" loosely for the sake of simplicity. Clearly, for different values of f the query-result schema changes, e.g., consider Example 5.3 with z removed from the head. However, this only affects the number of additional 0s in each of the n^3 answer tuples, therefore, it does not impact our construction.)

For the reduction from 3SUM, weights are assigned to the attribute values as $w_x(i) = A[i]$, $w_y(i) = B[i]$, $w_z(i) = C[i]$, $i \in [1, n]$, and $w_u(0) = 0$ for all other attributes u. By our weight assignment, the weights of the answers are $A[i] + B[j] + C[k]$, $i, j, k \in [1, n]$, and thus in one-to-one correspondence with the possible value combinations in the 3SUM problem. We first perform the preprocessing for direct access in $O(n^{2-\epsilon})$, which enables direct access to any position in the sorted array of query answers in $O(n^{2-\epsilon})$. By Lemma 5.6, weight lookup for a query result with zero weight is possible in $O(n^{2-\epsilon} \log n)$. Thus, we answer the original 3SUM problem in $O(n^{2-\epsilon'})$ for any $0 < \epsilon' < \epsilon$, violating the 3SUM hypothesis. □

For queries that do not have three independent free variables, we need a slightly different construction. We show next that two variables are sufficient to encode partial 3SUM solutions (i.e., pairs of elements), enabling a full solution of 3SUM via weight lookups. This yields a weaker lower bound than Lemma 5.7, but still is sufficient to prove intractability according to our yardstick.

LEMMA 5.8. *If a CQ Q is self-join-free and $\alpha_{free}(Q) = 2$, then direct access by SUM is not possible in $\langle n^{2-\epsilon}, n^{1-\epsilon}\rangle$ for any $\epsilon > 0$ assuming 3SUM.*

PROOF. We show that a counterexample query would violate the 3SUM hypothesis. Let A, B, and C be three integer arrays of a 3SUM instance of size n. We construct a database instance with

Query condition	Direct access	Complexity	Reason
acyclic $\alpha_{\text{free}}(Q) = 1$	possible in	$\langle n \log n, 1\rangle$	Lemma 5.9
acyclic $\alpha_{\text{free}}(Q) = 2$	not possible in	$\langle n^{2-\epsilon}, n^{1-\epsilon}\rangle$	3SUM
acyclic $\alpha_{\text{free}}(Q) \geq 3$	not possible in	$\langle n^{2-\epsilon}, n^{2-\epsilon}\rangle$	3SUM
cyclic	not possible in	$\langle n \text{ polylog } n, \text{polylog } n\rangle$	HYPERCLIQUE

Fig. 8. Possibility of direct access by sum of weights for acyclic self-join-free conjunctive queries.

attribute weights like in the proof of Lemma 5.7, but now with only two free and independent variables x and y. Hence the weights of the n^2 query results are in one-to-one correspondence with the corresponding sums $A[i] + B[j]$, $i, j \in [1, n]$. We run the preprocessing phase for direct access in $O(n^{2-\epsilon})$, which allows us to access the sorted array of query results in $O(n^{1-\epsilon})$. For each value $C[k]$ in C, we perform a weight lookup on Q for weight $-C[k]$, which takes time $O(n^{1-\epsilon} \log n)$ (Lemma 5.6). If that returns a valid index, then there exists a pair (i, j) of A and B with sum $A[i] + B[j] = -C[k]$, which implies $A[i] + B[j] + C[k] = 0$; otherwise, no such pair exists. Since there are n values in C, total time complexity is $O(n \cdot n^{1-\epsilon} \log n) = O(n^{2-\epsilon} \log n)$. This procedure solves 3SUM in $O(n^{2-\epsilon'})$ for any $0 < \epsilon' < \epsilon$, violating the 3SUM hypothesis. □

A special case of Lemma 5.8 is closely related to the problem of selection in $X + Y$ [30], where we want to access the k-th smallest sum of pairs between two sets X and Y. This is equivalent to accessing the answers to $Q_{XY}(x, y) :- R(x), S(y)$ by a SUM order. It has been shown that if X and Y are given sorted, then selection (single access) is possible even in *linear* time [20, 35]. Thus, for Q_{XY} direct access by SUM is possible in $\langle n \log n, n\rangle$ if we sort the relations during the preprocessing phase. Compared to our $\langle n^{2-\epsilon}, 1 - \epsilon\rangle$ lower bound (see also Figure 8), notice that even though the preprocessing of this algorithm is lower (asymptotically), the access time is not sublinear ($epsilon = 0$).

So far, we have covered all self-join-free CQs with $\alpha_{\text{free}}(Q) > 1$, which, by Lemma 5.4, proves the negative part of Theorem 5.1. Next, we show that the remaining acyclic CQs (those with $\alpha_{\text{free}}(Q) \leq 1$ or equivalently, an atom containing all the free variables) are tractable. For these queries, a single relation contains all the answers, so direct access can easily be supported by reducing, projecting, and sorting that relation.

LEMMA 5.9. *If a CQ Q is acyclic and an atom contains all the free variables, then direct access by SUM is possible in $\langle n \log n, 1\rangle$.*

PROOF. Since all free variables appear in one atom $R_f(\mathbf{X}_f)$, we can apply a linear-time semi-join reduction as in the Yannakakis algorithm [46] to remove the dangling tuples and then compute the query answers by projecting R on the free variables. Then, we sort the query answers by the sum of weights, which takes total time $O(n \log n)$ for preprocessing. We maintain the sorted answers in an array, which enables constant-time direct access to individual answers in ranked order. □

We now combine these lemmas with the fact that Boolean self-join-free cyclic CQs cannot be answered in $O(n \text{ polylog } n)$ time assuming HYPERCLIQUE, completing the proof of Theorem 5.1.

6 SELECTION BY LEXICOGRAPHIC ORDERS

We next investigate the tractability of a simpler version of the problem: When is *selection*, i.e., direct access to a *single* query answer, possible in quasilinear time? In this section, we answer this question for lexicographic orders, and in Section 7, we move to the case of SUM. Unlike direct-access, we show that selection can be efficiently achieved for any lexicographic order, as long as the query is free-connex. Our main result in this setting is summarized below:

Theorem 6.1 (Selection by LEX). *Let Q be a CQ and L be a partial lexicographic order.*

- *If Q is free-connex, then selection by L is possible in $\langle 1, n\rangle$.*
- *Otherwise, if Q is also self-join-free, then selection by L is not possible in $\langle 1, n \operatorname{polylog} n\rangle$, assuming Seth and Hyperclique.*

Our theorem shows that when we limit ourselves to the problem of selection, the tractability of the problem depends only on the query structure and is independent of the lexicographic order.

Example 6.2. Recall that direct access by L is intractable for $Q(v_1, v_2, v_3) :- R(v_1, v_3), S(v_3, v_2)$ with L being the lexicographic order $\langle v_1, v_2, v_3\rangle$ or the partial lexicographic order $\langle v_1, v_2\rangle$. The former contains a disruptive trio, while the latter is not L-connex. However, selection is tractable in both cases. Still, if we project out the middle variable v_3 and the head of the CQ is $Q(v_1, v_2)$, then the CQ is not free-connex and thus, selection becomes intractable for any lexicographic order.

For the negative part of Theorem 6.1, we reduce the problem of selection to that of counting query answers.

Lemma 6.3. *For a CQ Q, if selection by some ranking function is possible in $\langle 1, f(n)\rangle$ for a function f, then counting the answers to the CQ Q is possible in $O(f(n) \log n)$.*

Proof. We reduce the counting problem to the selection problem under any ranking function. If the number of relations in the query is ℓ, then an upper bound on the number of answers is n^ℓ. We use selection to determine whether any index contains an answer. With binary search on the range of indices $[0, n^\ell)$, we can find the smallest index that does not correspond to an answer. This process requires only $O(\log n^\ell) = O(\log n)$ selections, since ℓ is constant. □

We can now exploit lower bounds based on Seth. The proof does not rely on the properties of lexicographic orders and thus captures any possible ordering of the query answers.

Lemma 6.4. *If a self-join-free CQ Q is not free-connex, then selection by any ranking function is not possible in $\langle 1, n \operatorname{polylog} n\rangle$ assuming Seth and Hyperclique.*

Proof. We use the fact that, assuming Seth, the answers to a self-join-free and acyclic non-free-connex CQ cannot be counted in $O(n^{2-\epsilon})$ for any constant ϵ [34]. By Lemma 6.3, if selection is possible in $O(n \operatorname{polylog} n)$, then we can also count the number of query answers in $O(n \operatorname{polylog} n)$, contradicting our hypothesis. Cyclic CQs are covered by the hardness of Boolean self-join-free cyclic CQs based on Hyperclique, completing the proof. □

For the remainder of this section, we give a selection algorithm that together with Lemma 6.4 completes the proof of Theorem 6.1

6.1 Lexicographic Selection Algorithm

We first claim that, for any free variable in a free-connex CQ, we can efficiently compute the histogram of its assignments in the query answers. This is essentially equivalent to a group-by query that groups the query answers based on a single variable and then counts how many answers fall within each group.

Lemma 6.5. *Let Q be a free-connex CQ and $v \in \texttt{free}(Q)$. Given an input database I, we can compute in linear time how many answers in $Q(I)$ assign c to v for each value c in the active domain of v.*

Proof. Following Proposition 2.3, we can transform the problem to an equivalent problem with a full acyclic CQ Q'. We then take a join-tree for Q', identify a node V_p containing v, and introduce

a new node V_r as a neighbor of V_p. We associate V_r with the single variable v, assign V_r with a unary relation that contains the active domain of v, and set V_r to be the root of the tree. Then, we follow the preprocessing explained in Section 3.1 over this tree. By the end of this preprocessing, each tuple is given a weight that indicates the number of different answers that this tuple agrees with when only joining its subtree. Thus, the weights for V_r will contain the desired values. □

This count guides our selection algorithm, as it iteratively chooses an assignment for the next variable in the lexicographic order. Comparing the desired index with the count, it chooses an appropriate value for the next variable, filters the remaining relations according to the chosen value, and continues with the next variable.

Lemma 6.6. *Let Q be a free-connex CQ and L be a partial lexicographic order. Then, selection by L is possible in $\langle 1, n\rangle$.*

Proof. Let $\langle v_1, \ldots, v_m\rangle$ be a completion of L to a full lexicographic order, and let k be the index we want to access. We perform the following starting with $i = 1$: Let $c_1, \ldots, c_m$ be the ordered values in the active domain of v_i (the algorithm does not sort them, because that would already take $O(n \log n)$). We use Lemma 6.5 to count, for each c_r, the number of answers that assign c_r to v_i, denoted by $\texttt{weight}(c_r)$. Then, we find j such that $\sum_{r=1}^{j-1} \texttt{weight}(c_r) \leq k < \sum_{r=1}^{j} \texttt{weight}(c_r)$ and select the value c_j for v_i. This computation can be done in $O(n)$ without sorting if we use a weighted selection algorithm [30]. We proceed to filter all relations according to the $v_i = c_j$ assignment, update k to $k - \sum_{r=1}^{j-1} \texttt{weight}(c_r)$, and continue iteratively with $i + 1$. In each iteration, the value for another variable is determined, where $\sum_{r=1}^{j-1} \texttt{weight}(c_r)$ answers contain a strictly smaller value for the variable, and the next iterations break the tie between the $\texttt{weight}(c_j)$ answers that have this value. For the running time, each iteration takes linear time and we have a constant number of iterations (one iteration for every free variable). □

7 SELECTION BY SUM OF WEIGHTS

We now move on to the problem of selection by SUM order. Given that direct access by this order with quasilinear preprocessing and polylogarithmic delay is possible only in very few cases, it is a natural question to ask how the tractability landscape changes when considering the simpler task of *selection*.

7.1 Overview of Results

We show that the simplifications move only a narrow class of queries to the tractable side. For example, the two-path query $Q_2(x, y, z) :- R(x, y), S(y, z)$ is tractable for selection, even though it is not for direct access. However, the three-path query $Q_3(x, y, z, u) :- R(x, y), S(y, z), T(z, u)$ remains intractable. Given that Q_2 and Q_3 both have two free and independent variables, a different criterion than that of Section 5 ($\alpha_{\text{free}}(Q)$ or number of atoms containing the free variables) is needed for classification. To this end, we use hypergraph $\mathcal{H}_{\texttt{free}}(Q)$. Recall that it is the restriction of the query hypergraph $\mathcal{H}(Q)$ to the free variables, i.e., all the other variables are removed.

Definition 7.1 (Maximal Hyperedges). For a hypergraph $\mathcal{H} = (V, E)$, we denote the *number of maximal hyperedges* w.r.t. containment by $\text{mh}(\mathcal{H})$, i.e., $\text{mh}(\mathcal{H}) = |\{e \in E \mid \nexists e' \in E : e \subset e'\}|$. The number of maximal hyperedges of a query Q is $\text{mh}(Q) = \text{mh}(\mathcal{H}(Q))$ and the number of free-maximal hyperedges of Q is $\text{fmh}(Q) = \text{mh}(\mathcal{H}_{\texttt{free}}(Q))$.

Example 7.2. For $Q(x, z, w) :- R(x, y), S(y, z), T(z, w), U(x)$, we have $\text{mh}(Q) = 3$, because U is contained in R, and $\text{fmh}(Q) = 2$, because after removing the existentially-quantified y, the remainder of the S-hyperedge is contained in T.

Remark 4. For any CQ Q, we have $\alpha_{\text{free}}(Q) \leq \text{fmh}(Q)$. This follows from the fact that each independent variable must appear in a maximal hyperedge and that each hyperedge cannot contain more than one independent variable by definition. Note also that the condition $\alpha_{\text{free}}(Q) \leq 1$ is equivalent to $\text{fmh}(Q) \leq 1$, giving us a third possible way to express the criterion of Theorem 5.1 for direct access.

We summarize the results of this section in the following theorem, which classifies CQs Q based on $\text{fmh}(Q)$:

Theorem 7.3 (Selection by SUM). *Let Q be a CQ.*

- *If Q is free-connex and $\text{fmh}(Q) \leq 2$, then selection by SUM is possible in $\langle 1, n \log n \rangle$.*
- *Otherwise, if Q is also self-join-free, then selection by SUM is not possible in $\langle 1, n \,\text{polylog}\, n \rangle$. assuming 3sum, Hyperclique, and Seth.*

Example 7.4. For the query $Q_2(x, y, z) :- R(x, y), S(y, z)$, we have already shown in Section 5 that direct access by SUM is intractable. However, given that it has two maximal hyperedges, only one access (or a constant number of them) is in fact possible in $O(n \log n)$. The situation does not change for $Q_3'(x, y, z) :- R(x, y), S(y, z), T(z, u)$, because the hyperedge of T is contained in S in the free-restricted hypergraph. However, $Q_3(x, y, z, u) :- R(x, y), S(y, z), T(z, u)$, which keeps the variable u in the answers is intractable for selection, because now T corresponds to a free-maximal hyperedge.

Before proving Theorem 7.3, we first introduce some necessary concepts and prove a useful lemma.

Absorbed atoms and variables. We say that an atom e is *absorbed* by an atom $e' \neq e$ if $V \subseteq V'$ where V and V' are their sets of variables, respectively. Additionally, we say that a variable v is absorbed by a variable $u \neq v$ if (1) they appear in exactly the same atoms and (2) it is not the case that v is free and u is not free. As evident from Theorem 7.3, adding to a query atoms or variables that are absorbed by existing ones does not affect the complexity of selection. We prove this claim first and use it later in our analysis to treat queries that contain absorbed atoms or variables.

Definition 7.5 (Maximal Contraction). A query Q' is a *contraction* of Q if we can obtain Q' by iteratively removing absorbed atoms and variables, one at a time. Q^m is a *maximal contraction* of Q if it is a contraction and there is no contraction of Q^m.

Note that the number of atoms of a maximal contraction Q^m of Q is $\text{mh}(Q)$.

Example 7.6. Consider $Q(x, y, z) :- R(x, u, y), S(y), T(y, z), U(x, u, y)$. Here, $S(y)$ is absorbed by $R(x, u, y)$ and $U(x, u, y)$, and the latter two absorb each other. Additionally, the free variable x absorbs u, since these two variables appear together in R and U. Thus, a maximal contraction of Q is $Q^m(x, y, z) :- R(x, y), T(y, z)$, which is unique up to renaming. The number of maximal hyperdges of Q is $\text{mh}(Q) = 2$.

Lemma 7.7. *Selection for a CQ Q by SUM is possible in $\langle 1, g(n) \rangle$ if selection for a maximal contraction Q^m of Q by SUM is possible in $\langle 1, g(n) \rangle$. The converse is also true if Q is self-join-free.*

Proof. For the "if" direction, we use selection on Q^m to solve selection on Q. We can remove absorbed atoms from Q after making sure that the tuples in the database satisfy those atoms. Thus, to remove an atom $S(\mathbf{Y})$ that is absorbed by $R(\mathbf{X})$, we filter the relation R based on the tuples of S. To remove a variable v that is absorbed by u, in all relations that contain both u and v, we "pack" them together: We remove v and replace the u-values by values that represent the pair (u, v) and

assign to it the weight $w(u) + w(v)$. (Note that we assign $w(z) = 0$ for all variables z that are not free.) After separating any packed variables, Q^m over the modified database has the same answers as Q over the original one and the weights are preserved.

For the "only if" direction, we create an extended database where the answers to Q are the same as those of Q^m over the original database. For each step of the contraction, we make a modification of the database. If an atom $S(\mathbf{Y})$ was removed because it was absorbed by another atom $R(\mathbf{X})$, then we create the relation S by copying $\pi_{\mathbf{Y}}(R)$. Note that we are allowed to create S without restrictions because Q has no self-joins, hence the database does not already contain the relation. If a variable v was removed because it was absorbed by another variable u, then we extend all the relations that u appears in with another attribute v that takes the constant $\perp$ value everywhere and has weight $w_v(\perp) = 0$. After projecting away the new variable, this construction does not change the query answers or their weights.

The above reductions take linear time, which is dominated by $g(n)$, since $g(n)$ is trivially in $\Omega(n)$ for the selection problem. □

To prove Theorem 7.3, we first limit our attention to the class of full CQs (for them $\text{mh}(Q) = \text{fmh}(Q)$) and prove the positive part in Section 7.2 and the negative part in Section 7.3. We then extend those results to more general CQs with projections in Section 7.4.

7.2 Tractability Proofs for Full CQs

In this section, we provide tractability results for full CQs with $\text{mh}(Q) \leq 2$. First, we consider the trivial case of $\text{mh}(Q) = 1$, where the maximal contraction of Q has only one atom. The lemma below is a direct consequence of the linear-time array selection algorithm of Blum et al. [9].

Lemma 7.8. *For a full CQ Q with $mh(Q) = 1$, selection by SUM is possible in $\langle 1, n\rangle$.*

Proof. By Lemma 7.7, it suffices to solve selection on the query $Q(x) :- R(x)$, which is a maximal contraction of all queries with $\text{mh}(Q) = 1$, up to renaming. Trivially, the weights of the single attribute can also be viewed as tuple weights. Thus, applying linear-time selection [9] on the tuples of R gives us the k-th smallest query answer. □

For the $\text{mh}(Q) = 2$ case, we rely on an algorithm by Frederickson and Johnson [20], which generalizes selection on the X+Y problem. If the two sets X and Y are given sorted, then the pairwise sums can be represented as a sorted matrix. A *sorted matrix* M contains a sequence of non-decreasing elements in every row and every column. For the $X + Y$ problem, a cell $M[i, j]$ contains the sum $X[i] + Y[j]$. Even though the matrix M has quadratically many cells, there is no need to construct it in advance, given that we can compute each cell in constant time. Selection on a union of such matrices $\{M_1, \ldots, M_\ell\}$ asks for the k-th smallest element among the cells of all matrices.

Theorem 7.9 ([20]). *Selection on a union of sorted matrices $\{M_1, \ldots, M_\ell\}$, where M_m has dimension $p_m \times q_m$ with $p_m \geq q_m$, is possible in time $O(\sum_{m=1}^{\ell} q_m \log(2p_m/q_m))$.*

Leveraging this algorithm, we provide our next positive result:

Lemma 7.10. *For a full CQ Q with $mh(Q) = 2$, selection by SUM is possible in $\langle 1, n \log n\rangle$.*

Proof. The maximal contraction of full CQs with $\text{mh}(Q) = 2$ is $Q_1(x, z) :- R(x), S(z)$ or $Q_2(x, y, z) :- R(x, y), S(y, z)$, up to renaming. Thus, by Lemma 7.7, it is enough to prove an $O(n \log n)$ bound for these two queries. As before, we turn the attribute weights into tuple weights. For Q_1, the attribute weights are trivially tuple weights and for Q_2, we assign each attribute weight to only one relation to avoid double-counting. Thus, for Q_2, we compute $w(r) = w_x(r[x]) + w_y(r[y])$ and $w(s) = w_z(s[z])$ for all $r \in R$ and $s \in S$, respectively. Since the query is

full, the weights of the query answers are in one-to-one correspondence with the pairwise sums of weights of tuples from R and S.

For Q_2, we group the R and S tuples by their y values: We create ℓ buckets of tuples where all tuples t within a bucket have equal $t[y]$ values. This can be done in linear time. For Q_1, we place all tuples in a single bucket. For each assignment of a y value (no assignment for the case of Q_1), the query answers with those values are formed by the Cartesian product of R and S tuples inside that bucket. Also, if the size of bucket m is n_m, then $n_1 + \cdots + n_\ell = |R| + |S| = O(n)$. We sort the tuples in each bucket (separately for each relation) according to their weight in $O(n \log n)$ time. Assume R_m and S_m are the partitions of R and S in bucket m, and $R_m[i]$ denotes the i-th tuple of R_m in sorted order (equivalently for $S_m[j]$). We define a union of sorted matrices $\{M_1, \ldots, M_\ell\}$ by setting for each bucket m: $M_m[i,j] = w(R_m[i]) + w(S_m[j])$ if $|R_m| \geq |S_m|$ or $M_m[i,j] = w(S_m[i]) + w(R_m[j])$ otherwise (this distinction is needed simply to conform with the way Theorem 7.9 is stated). Selection on these matrices is equivalent to selection on the query answers of Q. By Theorem 7.9, if matrix M_m has dimension $p_m \times q_m$ with $p_m \geq q_m$, then we can achieve selection in $O(\sum_{m=1}^{\ell} q_m \log(2p_m/q_m)) \subseteq O(\sum_{m=1}^{\ell} q_m \cdot 2p_m/q_m) = O(\sum_{m=1}^{\ell} p_m) = O(\sum_{m=1}^{\ell} n_m) = O(n)$. Overall, the time spent is $O(n \log n)$ because of sorting. □

7.3 Intractability Proofs for Full CQs

Though selection is a special case of direct access, we show that for most full CQs, time $O(n \operatorname{polylog} n)$ is still unattainable. We start from the cases covered by Lemma 5.7. To extend that result to the selection problem, note that a selection algorithm can be repeatedly applied for solving direct access. For queries with three free and independent variables, an $O(n^{2-\epsilon})$ selection algorithm would imply a $\langle 1, n^{2-\epsilon} \rangle$ direct-access algorithm, which we showed to be impossible. Therefore, the following immediately follows from Lemma 5.7:

Corollary 7.11. *If a full CQ Q is self-join-free and $\alpha_{free}(Q) \geq 3$, then selection by SUM is not possible in $\langle 1, n^{2-\epsilon} \rangle$ for any $\epsilon > 0$ assuming 3sum.*

This leaves only a small fraction of full acyclic CQs to be covered: queries with two or fewer independent variables and three or more maximal hyperedges. We next show that these queries all contain a length-3 chordless path,[12] a property that we will use to prove a lower bound.

Lemma 7.12. *The hypergraph of the maximal contraction of any full acyclic CQ with $\alpha_{free}(Q) < 3$ and $mh(Q) > 2$ contains a chordless path of four variables.*

Proof. First, for $\alpha_{\text{free}}(Q) = 1$, we have by Lemma 5.4 that an atom contains all free variables, thus $\text{mh}(Q) = 1$. For the case of $\alpha_{\text{free}}(Q) = 2$, let x, y be independent variables. We distinguish two cases.

The first case is that each of x and y appear in exactly one maximal hyperedge. Denote the maximal hyperedge containing y by e_y and the maximal hyperedge containing x by e_x. Since there are at least three maximal edges, there is a hyperedge e_0 such that $x, y \notin e_0$. Since e_0 is not absorbed by e_y, there exists $a \in e_0$ and $a \notin e_y$. Thus, a and y are not neighbors. Since $\{a, x, y\}$ is not an independent set, we conclude that a and x are neighbors, and because only one maximal hyperedge can contain x, we get that $a \in e_x$. Similarly, since e_0 is not absorbed by e_x, we conclude that there exists a neighbor b of y such that $b \in e_0$, $b \notin e_x$. Overall, we have a chordless path $x - a - b - y$.

[12] The conference version of this article [13] erroneously claims that the maximal contraction of these CQs is the three-path query $Q(x, y, z, u) :- R(x, y), S(y, z), T(z, u)$. This is not correct, because, for example, $Q'(x, y, z, u, b) :- R(x, b, y), S(y, b, z), T(z, b, u)$ also satisfies $\alpha_{\text{free}}(Q') < 3$ and $\text{mh}(Q') > 2$. However, as we show here, the same reduction that was used for the three-path query Q can also work for any of these CQs.

The second case is that x or y (or both) appear in at least two hyperedges. Assume w.l.o.g. that two maximal hyperedges contain x. According to the claim we shall prove next, this means that there exist non-neighbors a,b that are both neighbors of x. Then, since $\{a, b, y\}$ is not an independent set, we have that y is a neighbor of a or b. Assume w.l.o.g. it is a. Then, we have a path $y - a - x - b$ such that y and x are not neighbors and a and b are not neighbors. We conclude that also y and b are not neighbors, otherwise this path is a chordless cycle contradicting acyclicity.

We now prove the claim that if two maximal hyperedges in an acyclic hypergraph both contain a node x, then there exists a chordless path $a - x - b$ for some vertices a, b. If there are two non-neighboring neighbors of x, then we are done. For the sake of contradiction, assume that there are no such neighbors. We prove with induction on the number of neighbors that for every k neighbors of x, there exists a hyperedge that contains all of them and x. For the base of the induction, consider two neighbors of x. By our assumption, they must necessarily be neighbors. Since the graph is acyclic, every triangle must be covered by a hyperedge, so there exists a hyperedge containing both of them and x. For the inductive step, consider k neighbors of x. By the induction hypothesis, every subset of these neighbors of size $k-1$ appears in a hyperedge together with x. If there is no hyperedge that contains all k of them, then these neighbors (without x) form a $(k, k-1)$-hyperclique contradicting acyclicity. If there is, then these k neighbors along with x form a $(k+1, k)$-hyperclique. This contradicts acyclicity unless there is a hyperedge containing all k variables and x. This concludes the induction. Now consider all neighbors of x. By the induction, one edge contains all of them and x. This contradicts the fact that x appears in two maximal hyperedges. □

Now that we established the precise form of the queries we want to classify, we proceed to prove their intractability. We approach this in a different way than the other hardness proofs: Instead of relying on the 3SUM hypothesis, we instead show that tractable selection would lead to unattainable bounds for Boolean cyclic queries.

LEMMA 7.13. *If a full CQ Q is self-join-free and the hypergraph of its maximal contraction contains a chordless path of four variables, then selection by SUM is not possible in $\langle 1, n \operatorname{polylog} n \rangle$ assuming HYPERCLIQUE.*

PROOF. We will show that if selection for Q can be done in $O(n \operatorname{polylog} n)$, then the Boolean triangle query can be evaluated in the same time bound, which contradicts the HYPERCLIQUE hypothesis. Let $Q_\triangle() :- R'(x', y'), S'(y', z'), T'(z', x')$ be a query over a database I' of size $O(n)$. We will construct a database I for Q so weight lookup (see Definition 5.5) for Q over I will allow us to answer $Q_\triangle$ over I'.

Let $x - y - z - u$ be the chordless path in the hypergraph of the maximal contraction of Q. This implies that there are atoms $R(x, y, \mathbf{X}_R), S(y, z, \mathbf{X}_S), T(z, u, \mathbf{X}_T)$ in Q. We construct a database I where in all relations, we let x and u, respectively, take all the values that x' can take in I'. We repeat the same for y with y' and z with z'. The values that all the other variables can take in I are set to a fixed domain value $\perp$. Because the $x - y - z - u$ path is chordless, we know that the pair $x - z$ never appears in a single atom, and so is the case for $x - u$ and $y - u$. Therefore, the size of each relation in I is bounded by $|R'|$ or $|S'|$ or $|T'|$ and the size of I is thus $O(n)$. Now consider a query answer $q \in Q(I)$. If $\pi_u(q) = \pi_x(q)$, then $\pi_{xyz}(q)$ has to satisfy all three atoms of $Q_\triangle$. This is because an (x, y) pair of values has to satisfy $\pi_{xy}(R)$, which contains precisely the tuples of R', and similarly for (y, z) with $\pi_{yz}(S)$. For a (z, x) pair of values, these are the same as (z, u) and satisfy $\pi_{zu}(T)$ contains precisely the tuples of T'.

We now assign weights as follows: If $\mathtt{dom} \subseteq \mathbb{R}$, then $w_x(i) = i, w_u(i) = -i$, and for all other variables t, $w_t(i) = 0$. Otherwise, it is also easy to assign w_x and w_u in a way s.t. $w_x(i) = w_x(j)$ if and only if $i = j$ and $w_u(i) = -w_x(i)$. This is done by maintaining a lookup table for all the

domain values that we map to some arbitrary real number. Then, we perform weight lookup for Q to identify if a query result with zero weight exists. If it does for some result q, then $w_x(\pi_x(q)) + \ldots + w_u(\pi_u(u)) = 0$, hence, $\pi_x(q) = \pi_u(q)$ and $Q_\triangle$ is true, otherwise, it is false. If the time to access the sorted array of Q-answers takes $O(n \operatorname{polylog} n)$, then by Lemma 5.6 weight lookup also takes $O(n \operatorname{polylog} n)$, contradicting Hyperclique. □

For full CQs, the negative part of Theorem 7.3 for acyclic queries is proved by combining Corollary 7.11 and Lemma 7.13 together with Lemma 7.12 and Lemma 7.7, which show that we cover all queries. For self-join-free cyclic CQs, we once again resort to the hardness of their Boolean version based on Hyperclique.

So far, we have proved the restriction of Theorem 7.3 to full CQs as summarized below.

Lemma 7.14. *Let Q be a full CQ.*

- *If $mh(Q) \leq 2$, then selection by SUM is possible in $\langle 1, n \log n \rangle$.*
- *Otherwise, if Q is also self-join-free, then selection by SUM is not possible in $\langle 1, n \operatorname{polylog} n \rangle$. assuming 3sum and Hyperclique.*

7.4 CQs with Projections

To complete the proof of Theorem 7.3, we now show how the results from Sections 7.2 and 7.3 generalize to free-connex CQs. For that purpose, we mainly rely on Proposition 2.3, which allows us to reduce a free-connex CQ to a full CQ. One difficulty that we encounter is that the full CQ that we obtain by this reduction may not necessarily be unique, as it depends on the join tree of the inclusion extension that we choose to use. For instance, the inclusion extension allows us to include unary hyperedges $\{x\}$ for all query variables x if these do not exist already. Thus, the following definition will be useful:

Definition 7.15 (Reduced Full CQs). For a free-connex CQ Q, $\mathrm{RF}(Q)$ is the set of all possible full CQs we can obtain by the reduction of Proposition 2.3.

We now show that for our purposes, the tractability of a free-connex CQ Q coincides with that of any of the CQs in $\mathrm{RF}(Q)$.

Lemma 7.16. *Let Q be a free-connex CQ and $Q' \in RF(Q)$. Selection for Q by SUM is possible in $\langle 1, g(n) \rangle$ if selection for Q' by SUM is possible in $\langle 1, g(n) \rangle$. The converse is also true if Q is self-join-free.*

Proof. The "if" direction is trivial from Proposition 2.3: If Q is over a database I, then we can use Q' over a modified database I' to obtain exactly the same answers.

For the "only if" direction, we use selection on Q to answer selection on Q'. If Q' is over a database I', then we construct a modified database I for Q as follows: We copy all the relations of I' into I and for every existential variable of Q, we add an attribute to the corresponding relations that takes the same $\perp$ value in all tuples. The weight of all the new attributes is set to 0 for $\perp$. Now, the answers to Q over I are the same as those of Q' over I' if we ignore all the $\perp$ values from the answers.

The above reductions take linear time, which is dominated by $g(n)$ since $g(n)$ is trivially in $\Omega(n)$ for the selection problem. □

From Lemma 7.14, we can decide the tractability of any (self-join-free) full CQ in $\mathrm{RF}(Q)$ by the number of its maximal hyperedges. We now connect this measure to the free-maximal-hyperedges $\mathrm{fmh}(Q)$, which is a measure easily computable from the original query Q.

Lemma 7.17. *For a free-connex CQ Q, $fmh(Q) = mh(Q')$ for any $Q' \in RF(Q)$.*

Proof. Let T be the ext-$\mathtt{free}(Q)$-connex tree used to derive Q' and T' be the connected subtree containing exactly the free variables of Q (which are also the free variables of Q'). Recall that the nodes of T' correspond to the atoms of Q'.

First, we prove that every node of T' is a subset of some hyperedge of $\mathcal{H}_{\mathtt{free}}(Q)$ and as a result $\text{fmh}(Q) \leq \text{mh}(Q')$. Consider a node of T' and let V be the corresponding set of variables. Note that V are all free variables, since they appear in T'. Since T' is a subtree of T, which in turn is a join-tree of an inclusive extension of $\mathcal{H}(Q)$, there must exist an atom in Q that contains V. Let the set of variables of that atom be $V \cup V_1 \cup X$, for some disjoint sets V, V_1, X, where V_1 are free variables and X are existential. By the definition of $\mathcal{H}_{\mathtt{free}}(Q)$, there must exist a hyperedge $V \cup V_1$ in $\mathcal{H}_{\mathtt{free}}(Q)$.

Second, we prove that every maximal hyperedge in $\mathcal{H}_{\mathtt{free}}(Q)$ is a subset of some node of T' and as a result $\text{mh}(Q') \leq \text{fmh}(Q)$. Let V be the (free) variables of a hyperedge of $\mathcal{H}_{\mathtt{free}}(Q)$. Then, there must exist an atom e in Q that contains the V variables. If e corresponds to a node in T', then we are done. Otherwise, e corresponds to a node in $T \setminus T'$. Let V' be the first node of T' on the path from V to some node in T'. A path between V and any node of T' must necessarily pass through V', otherwise T would contain a cycle. Since V contains only free variables, each one of them has to appear in some node of T' and by the running intersection property, in all the nodes on the path to V. Therefore, V' contains all the variables of V. □

Lemmas 7.16 and 7.17 allow us to generalize the positive part of Lemma 7.14 from full CQs to free-connex CQs by simply replacing the $\text{mh}(Q)$ measure with $\text{fmh}(Q)$. For the negative part, we need three ingredients to cover the class of self-join-free CQs: The case of free-connex CQs with $\text{fmh}(Q) > 2$ is also covered by the lemmas above. For acyclic CQs that are not free-connex, we use Lemma 6.4, which applies to any ranking function, including SUM. Finally, the intractability of cyclic CQs follows from Hyperclique.

8 FUNCTIONAL DEPENDENCIES

In this section, we extend our results to apply to databases that are constrained by Functional Dependencies (FDs). From the point of view of a CQ, if the allowed input databases are restricted to satisfy an FD, then an assignment to some of the variables uniquely determines the assignment to another variable. Our positive results are not affected by this, since an algorithm can simply ignore the FDs. However, certain CQs that were previously intractable may now become tractable in the presence of FDs. Our goal is to investigate how the tractability landscape changes for all four variants we have investigated so far: direct access and selection by LEX or SUM orders.

Concepts and Notation for FDs. In this section, we assume that the database schema $\mathcal{S}$ is extended with FDs of the form $R : A \to B$, where A and B are sets of integers. This means that if the tuples of relation R agree on the attributes indexed by A, then they also agree on those indexed by B. Though FDs are usually defined directly on the schema of the database, we express them from now on using the query variables for convenience. More specifically, we assume that an FD has the form $R : \mathbf{X} \to \mathbf{Y}$ for some atom $R(\mathbf{Z})$ where $\mathbf{X}, \mathbf{Y} \subseteq \mathbf{Z}$. We now briefly explain why this assumption can be done without loss of generality. There are two factors that can render such a notation not well-defined. The first is self-joins. However, our negative results apply only to self-join-free CQs regardless of this issue, and positive results for self-join-free CQs naturally extend to CQs with self joins: Self-joins can be reduced in linear time to a self-join-free form by replacing the i-th occurrence of a relational symbol R with a fresh relational symbol R_i and then copying relation R into R_i. The second factor is repeated appearances of a variable in an atom (e.g., $R(x, x)$). Such an appearance can be eliminated in a linear-time preprocessing step that performs the selection on the relation and then removes the duplicate variable. For the other direction of the equivalence, the projected relation can be transformed to one with the repeated variable by duplicating the

relevant column in the relation. We say that an FD $\mathbf{X} \to \mathbf{Y}$ is *satisfied* by an input database I if for all tuples $t_1, t_2 \in R^I$, we have that if $t_1[x] = t_2[x], \forall x \in \mathbf{X}$, then $t_1[y] = t_2[y], \forall y \in \mathbf{Y}$. For such an FD, we sometimes say that $\mathbf{X}$ *implies* $\mathbf{Y}$. W.l.o.g., we assume that all FDs are of the form $R : \mathbf{X} \to y$, where y is a single variable, because we can replace an FD of the form $R : \mathbf{X} \to \mathbf{Y}$ with a set of FDs $\{R : \mathbf{X} \to y \mid y \in \mathbf{Y}\}$. If $|\mathbf{X}| = 1$, then we say that the FD is *unary*. In this article, we focus only on unary FDs.

Complexity and Reductions. In all previous sections, the computational problem we were considering was defined by a CQ and we were interested in the worst-case complexity over all possible input databases. Now, the problem is defined by a CQ Q and a set of FDs Δ. Thus, the input is limited to databases that satisfy all the FDs Δ. Therefore, to determine the hardness of a problem, we need to consider the complexity of the combination of CQ and FDs. In particular, the results in the previous sections apply when the given FD set is empty. Our reductions from now on are from a certain CQ and FD set to another CQ and another FD set.

Definition 8.1 (Exact Reduction). We say that there is an exact reduction from a CQ Q with FDs Δ to a CQ Q' with FDs Δ' if for every database I that satisfies Δ:

(1) We can construct a database I' in $O(|I|)$ that satisfies the FDs Δ'.
(2) There is a bijection τ from $Q'(I')$ to $Q(I)$ that is computable in $O(1)$.

Additionally, we say that an exact reduction is *weight-preserving* if for every weight function w given for Q, there is a weight function w' for Q' such that $w(\tau(q')) = w'(q'), \forall q' \in Q'(I')$.

Known Results. Carmeli and Kröll [12] reasoned about the complexity of enumerating the answers to a CQ by looking at an equivalent extended CQ over an extended schema and FDs. We recall the definition of this extension here[13] and then proceed to use it for our classification.

Definition 8.2 (FD-extension [12]). Given a self-join free CQ Q and a set of FDs Δ, we define two types of extension steps. For an FD $R : \mathbf{X} \to y$:

(1) If $\mathbf{X} \subseteq \mathbf{Z}$ for some atom $S(\mathbf{Z})$ and $y \notin \mathbf{Z}$, then increase the arity of S by one, replace $S(\mathbf{Z})$ with $S(\mathbf{Z}, y)$, and add $S : \mathbf{X} \to y$ to the FD set.
(2) If $\mathbf{X} \subseteq \texttt{free}(Q)$ and $y \notin \texttt{free}(Q)$, then add y to $\texttt{free}(Q)$.

The FD-extension of Q and Δ is a CQ Q^+ and a set of FDs Δ^+ that are obtained as the fixpoint of the above two extension steps.

Example 8.3. Consider the CQ $Q_{2P}(x, z) \coloneq R(x, y), S(y, z)$. This CQ is not free-connex, therefore selection (or direct access) is not possible by any order (Lemma 6.4). Now, if we know that the database satisfies the FD $S : y \to z$, then we can take the unique z-value for every y from S and add it to every tuple of R, while preserving the same query answers. Thus, we can extend the CQ to $Q^+_{2P}(x, z) \coloneq R(x, y, z), S(y, z)$ and add the FD $R : y \to z$. While the original CQ was not free-connex, notice that Q^+_{2P} now is. This makes it tractable for all the tasks that we consider in this article, since it is acyclic and R contains all the free variables (see Section 5).

Similarly, the FD-extension may transform a cyclic CQ into an acyclic one. For $Q_\triangle(x, y, z) \coloneq R(x, y), S(y, z), T(z, x)$ with the FD $S : y \to z$, we get the FD-extension $Q^+_\triangle(x, y, z) \coloneq R(x, y, z), S(y, z), T(z, x)$ with the additional FD $R : y \to z$, which is acyclic, because R contains all the variables. Like $Q^+_{2P}(x, z)$, we have that $Q^+_\triangle$ is tractable for all the tasks that we consider. This is despite the fact that the cyclic $Q_\triangle$ without FDs is intractable for all of these tasks.

[13] The original definition is more involved, since it also applies to self-joins. We only give the restriction of the definition required for our purposes. We also state it for general FDs instead of unary FDs, which is useful for our discussion in Section 8.3.

Previous work [12] showed exact reductions between the original CQ (with the original FDs) and its extension (with the extended FDs) in both directions, proving that the two tasks are essentially equivalent for the task of enumeration.

Theorem 8.4 ([12]). *Let Q be a self-join free CQ with FDs Δ. There are exact reductions between Q with Δ and Q^+ with Δ^+ in both directions.*

This theorem alone implies that, if Q^+ has a tractable structure (free-connex for enumeration), then the original CQ Q is tractable, too. That is because, according to the definition of an exact reduction, an instance for the extended schema can be built in linear time, and answers to the extension can be translated back to answers of the original CQ in constant time per answer.

We restate this result below in a slightly more general way, adding the fact that these reductions are actually weight-preserving. Intuitively, this means that any SUM ordering of the answers in one problem can be preserved through the reduction.

Lemma 8.5. *Let Q be a self-join free CQ with FDs Δ. There are weight-preserving exact reductions between Q with Δ and Q^+ with Δ^+ in both directions.*

Proof. The bijections between the answers in the exact reductions do not change the values of the free variables of Q. For the reduction from the query to its extension, we set the weights of the new free variables to zero. For the reduction from the extension to the original query, if the extension has a free variable y that is existential in the original query, then some free variable x in the original query implies y. To maintain the contribution of the y-weight in the query answers, we simply increase the weight of every domain value of x by adding the weight of the corresponding domain value of y (i.e., the one that is implied by the FD). □

Proving lower bounds using the extension is more tricky. If the extension has a structure that is known to make a CQ intractable (e.g., not free-connex for enumeration), then it is not necessarily the case that it is intractable together with the FDs. Still, Carmeli and Kröll [12] were able to prove lower bounds for the cases where the extension is not free-connex. We proceed to use the same technique that they used to prove a more general statement.

Eliminating FDs. Our goal is to show that for the extension of a self-join-free CQ, unary FDs essentially do not affect our classification. This result is similar in spirit to other results in database theory, where the complexities of problems with FDs were also identified to be the complexity of the original complexity criterion applied to the FD extension *after removing the FDs* [21, 24, 32]. To achieve that, we reduce a CQ without FDs to the same CQ with FDs, under the condition that the CQ and the FDs we reduce to are the extension of some CQ.

Lemma 8.6. *Let Q be a self-join free CQ with unary FDs Δ. There is a weight-preserving exact reduction from Q^+ without FDs to Q^+ over Δ^+.*

Proof. We are given a database instance I for Q^+ that does not necessarily satisfy any FDs, and we construct another database instance I' with roughly the same answers (there is a bijection τ such that $Q^+(I) = \tau(Q^+(I'))$) that satisfies the extended FDs Δ^+. Given a variable v, denote by $\mathbf{X}_v$ the set of all variables that are transitively implied by v. By the definition of the extension, for every variable v, every atom of Q^+ that contains v also contains all $\mathbf{X}_v$ variables. For every tuple in the relation that corresponds to such an atom, we replace the v-value with the concatenation of $\mathbf{X}_v$-values. We set the weight of a concatenated value to be equal to the weight of the v variable as it was in the original database I. This construction can be done in linear time, and the resulting database satisfies the FDs.

We now claim that this construction preserves the answers through a bijection. Note that for every free variable v, by the definition of the extension, the variables in $\mathbf{X}_v$ are all free. Every

answer to the original problem gives an answer to our construction by assigning every free variable v to the concatenation of the assignments of $\mathbf{X}_v$. Every answer to our construction gives an answer to the original problem by keeping only the value that corresponds to the original variable for every free variable. In both cases, the weights of the query answers are the same in the two instances. □

Example 8.7. We illustrate the reduction above through the CQ $Q(x, z, u) :- R(x, y), S(y, z), T(z, u)$ with FD $T : z \to u$. The FD-extension is $Q^+(x, z, u) :- R(x, y), S'(y, z, u), T(z, u)$ with extended FDs $T : z \to u$ and $S' : z \to u$. Notice that Q^+, like Q, is not free-connex. From that structural property of Q^+, we know that without any FDs it is intractable, e.g., for the task of selection by Lemma 7.16. The reduction shows that this is still the case, even if we take the extended FDs into account.

Concretely, the reduction takes a database that does not satisfy the extended FDs and constructs another database that does. For example, S' could contain tuples $(1, 1, 1)$ and $(1, 1, 2)$ where the middle attribute (corresponding to z) does not imply the third attribute (corresponding to u). To modify this database, we replace the z-values by values that pack z and u together as (z, u). Conceptually, and abusing the notation a little, $S'(y, z, u)$ is further replaced with $S''(y, (z, u), u)$ and the whole query with $Q^{+\prime}(x, (z, u), u) :- R(x, y), S''(y, (z, u), u), T'((z, u), u)$. The aforementioned two tuples of S' are thus replaced by $(1, (1, 1), 1)$ and $(1, (1, 2), 2)$. Now, the third attribute is trivially dependent on the middle one, because it is contained in the latter. This reduction is only possible because, being an FD-extension of Q, Q^+ always contains u in all of its atoms that contain z. The weights of the query answers are preserved if we only keep the weight of z in the concatenated (z, u) values, i.e., $w_z((1, 1)) = w_z(1)$ and $w_z((1, 2)) = w_z(1)$.

Note that a reduction in the opposite direction is trivial, since, given an instance that satisfies the FDs, it is also a valid instance for the instance without the FDs. Thus, this lemma proves that the two problems are equivalent. By combining this fact with the equivalence of CQs and their FD-extensions (Lemma 8.5), we obtain the following result that is also useful for lower bounds and allows us to classify queries based on the structure of their extension:

Theorem 8.8. *Let Q be a self-join free CQ with unary FDs Δ. There are weight-preserving exact reductions between Q over Δ and Q^+ without FDs in both directions.*

8.1 Sum of Weights

For a sum-of-weights order, applying a weight-preserving exact reduction cannot reorder the query answers, since their weight is preserved. More formally, the classes of self-join-free CQs that are tractable for selection and direct access are both *closed* under weight-preserving exact reductions. Therefore, Theorem 8.8 immediately proves that for both problems, we can classify self-join-free CQs by their FD-extension.

Theorem 8.9. *Let Q be a CQ with unary FDs Δ.*

- *If Q^+ is acyclic and an atom of Q^+ contains all the free variables, then direct access for Q by SUM is possible in $\langle n \log n, 1\rangle$.*
- *Otherwise, if Q is also self-join-free, then direct access for Q by SUM is not possible in $\langle n \operatorname{polylog} n, \operatorname{polylog} n\rangle$, assuming 3sum and Hyperclique.*

Theorem 8.10. *Let Q be a CQ with unary FDs Δ.*

- *If Q^+ is free-connex and $fmh(Q^+) \leq 2$, then selection for Q by SUM is possible in $\langle 1, n \log n\rangle$.*
- *Otherwise, if Q is also self-join-free, then selection for Q by SUM is not possible in $\langle 1, n \operatorname{polylog} n\rangle$, assuming 3sum, Hyperclique, and Seth.*

8.2 Lexicographic Orders

We now move on to lexicographic orders, where we also provide dichotomies for unary FDs. The analysis gets more intriguing compared to SUM, since the FDs may interact with the given lexicographic order in non-trivial ways that have not been explored by any previous work we know of. Like before, we use as our main tool an extension of the query according to the FDs and show an equivalence between the extension and the original problem with respect to tractability. The key difference now is that the extension may also *reorder* the variables in the lexicographic order according to the FDs.

8.2.1 Tractability Results. Our positive results rely on two ingredients. The first ingredient is to have a reduction that preserves precisely the same lexicographic order that we begin with. This is necessary, because if the query answers after the reduction follow a different lexicographic order, then we cannot conclude anything about the position of a query answer in the original problem. Thus, we define a stricter notion of an exact reduction.

Definition 8.11 (Lex-preserving Exact Reduction). Consider a CQ Q and FDs Δ, and a CQ Q' and FDs Δ' such that the free variables of Q are also free variables in Q'. An exact reduction via a bijection τ from Q and Δ to Q' and Δ' is called *lex-preserving* if, for every partial lexicographic order L for Q and for all query answers q_1, q_2 of Q', we have that $q_1 \preceq' q_2$ iff $\tau(q_1) \preceq \tau(q_2)$ where $\preceq$ and $\preceq'$ are orders implied by L.

In a lex-preserving exact reduction, we have the guarantee that even though the query answers might not be exactly the same, those that are in a 1-1 correspondence will be placed in the same position when ordered. Thus, such a reduction allows us to solve direct access or selection on a different problem and translate the answers back to the original problem. Analogously to weight-preserving exact reductions for SUM, our notions of tractability for lexicographic orders are preserved under lex-preserving exact reductions.

Now notice that the forward reduction from Q with Δ to Q^+ with Δ^+ in Lemma 8.5 is a lex-preserving exact reduction. This is because, as we argued before, the reduction to the extension does not change the values of any of the free variables. The following lemma is, similarly to Lemma 8.5, a slight generalization of the result of Carmeli and Kröll [12], this time suited to lexicographic orders.

Lemma 8.12. *Let Q be a self-join free CQ with FDs Δ. There is a lex-preserving exact reduction from Q with Δ to Q^+ with Δ^+.*

Importantly, the other direction of Lemma 8.12 is not true in general, since the FD-extension may contain additional free variables that do not appear in the original query Q. Similarly, the reduction in Lemma 8.6 is also not lex-preserving. As an example, consider the simple example of $Q(v_1, v_2, v_3) :- R(v_1, v_2, v_3)$ with the FD $R : v_1 \rightarrow v_3$. Sorting the constructed instance by $\langle v_1, v_2, v_3\rangle$ will actually result in an ordering according to $\langle (v_1, v_3), v_2, v_3\rangle$, which is the same as $\langle v_1, v_3, v_2\rangle$. That is precisely the reason why proving lower bounds for the case of lexicographic orders is not as straightforward as the SUM case.

Going back to the positive side, we can now use Lemma 8.12 to reduce our problem to its extension. But to cover all tractable cases, we need to modify the FD-extension so the FDs are also taken into account in the lexicographic order. In particular, if a variable u is implied by a variable v and u comes after v in the order, then u is placed right after v in the reordering.

Definition 8.13 (FD-reordered Extension). Given a self-join-free CQ Q, a set of unary FDs Δ, and a partial lexicographic order L, their FD-reordered extension is a CQ Q^+, a set of unary FDs Δ^+, and a partial lexicographic order L^+. We take Q^+ and Δ^+ to be as defined in Definition 8.2, and L^+

is obtained from L by applying the following reordering step iteratively from index $i = 0$ until we reach the end of the list L: Find all variables $\mathbf{X}_{L[i]}$ that are transitively implied by $L[i]$ and place them in consecutive indexes starting from $i + 1$. Note that L may grow in this process to contain variables that are free in Q^+ though they are not free in Q.

Example 8.14. Consider the CQ $Q(v_1, v_2, v_3, v_4) :- R(v_1, v_3), S(v_3, v_2), T(v_2, v_4)$ with the FD $R : v_1 \rightarrow v_3$ and the lexicographic order $L = \langle v_1, v_2, v_3, v_4 \rangle$. This order contains the disruptive trio v_1, v_2, v_3 and is intractable for direct access according to the results of Section 3. When applying the FD-extension, we get that $Q^+ = Q$, because v_1 already appears with v_3 in R. For the FD-reordered extension, we reorder the lexicographic order into $L = \langle v_1, v_3, v_2, v_4 \rangle$, which contains no disruptive trio and is tractable for direct access.

We next prove two important properties for the FD-reordered extension:

Lemma 8.15. *For every variable v of an order L^+ of an FD-reordered extension, all variables transitively implied by v that appear after v in L^+ have to appear consecutively after v in L^+.*

Proof. The property holds at the end of the process due to the transitivity of implication. Consider a variable y implied by v that appears after v in L^+ and assume there is a variable x that is in between v and y but is not implied by v. We can further assume that x is the first variable in the order with that property. Now, x must have been inserted at that position when handling another variable z that is in between v and x in L^+ and that $z \rightarrow x$. Since x is the first one not implied by v, we have that $v \rightarrow z$, which gives us $v \rightarrow x$, contradicting our assumption. □

We next show that the reordering of the variables gives the same result because of the extended FDs Δ^+. As a consequence, we can study the complexity of the query with the FD-reordering.

Lemma 8.16. *Given a self-join-free CQ Q, a set of unary FDs Δ, and a partial lexicographic order L, for every database I that satisfies Δ^+, ordering $Q^+(I)$ by L is the same as ordering it by L^+.*

Proof. Once the value for a variable v is set, a variable implied by v can have at most one possible value, so as long as it comes after v in the order, its exact position, or whether it is free cannot influence the answer ordering. Therefore, the reordered extension is equivalent to the original extension. □

We now have what we need to show that if the reordered extension has a tractable structure, then we can conclude tractability for the original problem. Specifically, if the CQ Q^+ of the FD-reordered extension is free-connex, L^+-connex, and has no disruptive trio, then we know it is tractable with respect to direct access by Theorem 4.1. Lemma 8.16 shows that direct access to the extension by L^+ is the same as direct access by L, which is the order that we want. Finally, we use the fact that the reduction given in Lemma 8.5 preserves lexicographic orders to conclude that, given an input to Q, we can construct an FD-reordered extension where tractable direct access for Q^+ by L^+ will give us tractable direct access for Q by L. Following the same process for selection and using Theorem 6.1, we conclude that if the CQ Q^+ is free-connex, then selection for Q by L is tractable.

8.2.2 Intractability Results. We begin by some negative results that can easily be inferred from the results we have already proved in this article or by past work.

Lemma 8.17. *Let Q be a self-join-free CQ with unary FDs Δ. If Q^+ is not free-connex, then for any ranking function, direct access for Q is not possible in $\langle n \operatorname{polylog} n, \operatorname{polylog} n \rangle$ assuming sparseBMM and Hyperclique, and selection for Q is not possible in $\langle 1, n \operatorname{polylog} n \rangle$ assuming Seth and Hyperclique.*

Proof. For direct access, the impossibility is implied by the hardness of enumeration. If we could have direct access for Q, then we can also have enumeration with the same time bounds. Then, by the exact reduction of Theorem 8.8, we would be able to enumerate the answers to the non-free-connex CQ Q^+ with no FDs with quasilinear preprocessing and polylogarithmic delay, which is known to contradict sparseBMM or Hyperclique [5, 11].

For selection, we use Lemma 6.3 together with the simple observation that an exact reduction preserves the number of query answers. Irrespective of the order, Theorem 8.8 tells us that counting the answers to Q with Δ gives us the count of query answers of Q^+ with no FDs. Since a logarithmic number of selections can be used to find the latter, the count of answers of the non-free-connex CQ Q^+ can be found in quasilinear time, contradicting Seth (if Q^+ is acyclic) or Hyperclique (if Q^+ is cyclic). □

It is left to handle the cases that the reordering of the extension is free-connex acyclic but has a disruptive trio or is not L^+-connex. The case that the extension is not L^+-connex can be shown using a reduction from enumeration for a non-free-connex CQ (Lemma 3.12), as we did in Section 4.2.

Lemma 8.18. *Let Q be a self-join-free CQ with unary FDs Δ, and let L be a partial lexicographic order. If Q^+ is acyclic but not L^+-connex, then direct access for Q by L is not possible in $\langle n \text{ polylog } n, \text{polylog } n\rangle$ assuming sparseBMM.*

Proof. Using Lemma 3.12, it is enough to find a prefix L' of L^+ that is closed under implication such that the extension Q^+ is not L'-connex. We simply take $L' = L^+$. Then, we conclude that there is efficient enumeration for the acyclic but not free-connex Q^+ with L' as free variables. Since L' is closed under implication, Q^+ is an extension of itself, so we can use Lemma 8.6 to conclude that there exists efficient enumeration to Q^+ with L' as free variables even without FDs. This is a contradiction, as it is not free-connex. □

The case where the reordered extension has a disruptive trio v_1, v_2, v_3 is slightly more intricate, as we cannot directly use Lemma 8.6. One might hope that, as we did in the proof of Lemma 3.13, we would be able to take a prefix of L^+ that ends in v_2 and then apply Lemma 8.6. However, that is not always possible here, because we have the additional restriction that the prefix we pick has to be closed under implication. This is required so the CQ we obtain when we restrict the free variables to the prefix is a valid FD-extension. Unfortunately, a prefix that includes v_2 but not v_3 and is closed under implication does not necessarily exist. That is the case when some variable implies both v_2 and v_3.

Example 8.19. Consider the CQ $Q(v_1, v_2) \coloneq R(v_1, v_3), S(v_3, v_2)$ with the FD $S : v_2 \to v_3$ and the lexicographic order $L = \langle v_1, v_2\rangle$. The extended reordering is $L^+ = \langle v_1, v_2, v_3\rangle$, which contains the disruptive trio v_1, v_2, v_3. To reuse our previous approach, we would want to claim that using lexicographic direct access to Q^+, we can enumerate the CQ with only v_1, v_2 as free variables (which happens to be in this case the same as Q that we started with). However, this is not a contradiction, because Q is not known to be hard for enumeration, as it has FDs and it is not an extension (v_2 implies v_3, while v_2 is free and v_3 is existential). In fact, we cannot find any prefix L' of L^+ that is closed under implication such that Q^+ is not L'-connex. To circumvent this issue, we encode the enumeration of Q without FDs into the extension by combining the binary search approach from Lemma 3.12 with the concatenation reduction from Lemma 8.6. The difference is that before we used binary search to enumerate a prefix of the free variables, and now this prefix might stop in the middle of a variable with concatenated values. Thus, v_2 will be assigned the values (v_2, v_3), and we will use binary search to skip over the v_3 values.

Lemma 8.20. *Let Q be a self-join-free CQ with unary FDs Δ, and let L be a partial lexicographic order. If Q^+ is acyclic and L^+ contains a disruptive trio in Q^+, then direct access for Q by L is not possible in $\langle n \text{ polylog } n, \text{polylog } n\rangle$ assuming sparseBMM.*

Proof. Consider a disruptive trio v_1, v_2, v_3 in Q^+ with respect to L^+. Let L' be the prefix of L^+ ending in v_2, and let Q' be the query with the body of Q^+ and the free variables L'. As v_1, v_3, v_2 is an L'-path, we know that Q' is acyclic but not free-connex, and so it cannot be enumerated efficiently without FDs assuming sparseBMM.

We first claim that the reordering L^+ is stable with respect to the first occurrences of implying variables. More precisely, let a and b be variables in L^+ such that a appears before b in L^+. We claim that the first variable implying b does not appear before the first variable implying a. Indeed, consider the first variable v_b implying b. If v_b appears before a, then by Lemma 8.15, v_b also implies a. So, the first variable that implies a is v_b or a variable before it. As a consequence, due to the reordering, the first variable implying a value that appears after v_2 appears after all first variables implying values before (and including) v_2.

We can now claim that we can enumerate the answers to Q' without FDs using lexicographic direct access to Q^+ with FDs. We use the same construction as we did in Lemma 8.6 by assigning each variable a concatenation of the variables it implies, except that now we need to be careful about the order in which we concatenate: We start with any variables in L^+, ordered by L^+. The constructed database satisfies the FDs. It is only left to use binary search, similarly to Lemma 3.12, to enumerate the distinct values of the variables of L'. Due to the previous paragraph, we know that these appear as a prefix, before the first value of a variable after v_2. □

The results of this section regarding lexicographic orders are summarized as follows:

Theorem 8.21. *Let Q be a CQ with unary FDs Δ and L be a partial lexicographic order.*

- *If Q^+ is free-connex and L^+-connex and does not have a disruptive trio with respect to L^+, then direct access for Q by L is possible in $\langle n \log n, \log n\rangle$.*
- *Otherwise, if Q is also self-join-free, then direct access for Q by L is not possible in $\langle n \text{ polylog } n, \text{polylog } n\rangle$, assuming sparseBMM and Hyperclique.*

Theorem 8.22. *Let Q be a CQ with unary FDs Δ and L be a partial lexicographic order.*

- *If Q^+ is free-connex, then selection for Q by L is possible in $\langle 1, n\rangle$.*
- *Otherwise, if Q is also self-join-free, then selection for Q by L is not possible in $\langle 1, n \text{ polylog } n\rangle$, assuming Seth and Hyperclique.*

8.3 A Note on General FDs

We discussed only unary FDs, where a single variable implies another. The positive side of our results also holds for general FDs where a combination of variables may imply a variable. We simply need to take the general form of the extension (given an FD $x_1, \ldots, x_m \rightarrow y$, we add y wherever all of $x_1, \ldots, x_m$ appear). If the extension has a tractable form, then Lemmas 8.5 and 8.12 show that the original query is tractable, too. However, extending the negative results requires a much more intricate analysis that goes beyond the scope of this work. Already for enumeration, even though Carmeli and Kröll [12] showed a classificiation for general FDs when the extension is acyclic, the cyclic case is not resolved, and they provide a specific example of a CQ and FDs where the complexity is unknown.

9 CONCLUSIONS

We investigated the task of constructing a direct-access data structure to the output of a query with an ordering over the answers, as well as the restriction of the problem to accessing a single

answer (the selection problem). We presented algorithms for fragments of the class of CQs for lexicographic and sum-of-weights orders. The direct access algorithms take quasilinear construction time in the size of the database and logarithmic time for access. For selection, our algorithms take quasilinear or even linear time. We further showed that within the class of CQs without self-joins, our algorithms cover all the cases where these complexity guarantees are feasible, assuming conventional hypotheses in the theory of fine-grained complexity. We were also able to precisely capture how the frontier of tractability changes under the presence of unary FDs.

This work opens up several directions for future work, including the generalization to more expressive queries (CQs with self-joins, union of CQs, negation, etc.), other kinds of orders (e.g., min/max over the tuple entries), and a continuum of complexity guarantees (beyond ⟨quasilinear, logarithmic time⟩).

Generalizing the question posed at the beginning of the Introduction, we view this work as part of a bigger challenge that continues the line of research on *factorized representations* in databases [36, 37]: How can we represent the output of a query in a way that, compared to the explicit representation, is fundamentally more compact and efficiently computable, yet equally useful to downstream operations?

APPENDIX

A NOMENCLATURE

Symbol	Definition
R, S, T, U, R_1, R_2	relation
e, V, V_1, V_2	atom/hyperedge/node of join tree
$\mathcal{S}$	schema
I	database (instance)
n	size of I (number of tuples)
$\mathtt{dom}$	domain
$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$	list of variables/attributes
t	tuple
x, y, z, u, v_1, v_2	variable
Q	CQ
$q \in Q(I)$	query answer of CQ Q over database I
$Q(I)$	set of answers of Q over I
$\pi_{\mathbf{X}}(R)$	projection of R on $\mathbf{X}$
$\mathbf{X}_f, \mathtt{free}(Q)$	free variables of query Q
$\mathtt{var}(Q), \mathtt{var}(e)$	variables of query or atom
$\mathtt{atoms}(Q)$	set of query atoms
$\mathcal{H}(Q) = (V, E)$	hypergraph associated with query Q
$\mathcal{H}_{\mathtt{free}}(Q) = (V, E)$	restriction of $\mathcal{H}(Q)$ to free variables only
T	join tree
$\mathcal{V}$	set of join tree nodes
$\alpha_{\text{free}}(Q)$	maximum number of independent free variables of Q
$\text{mh}(Q)$	number of maximal hyperedges (with respect to containment) in $\mathcal{H}(Q)$
$\text{fmh}(Q)$	number of free-maximal hyperedges = maximal hyperedges in $\mathcal{H}_{\mathtt{free}}(Q)$
$L = \langle v_1, \ldots, v_m \rangle$	lexicographic order of variables
w_x	weight function for variable x: $\mathtt{dom} \to \mathbb{R}$
w_Q	weight function for query answers
w	short form for all w_x and w_Q
Σw	sum-of-weights order
λ	a real-valued weight
$\preceq$	total order over query answers
Π	family of orders
order	a binary relation as in partial/total order
ordering	a sorted list according to an order
$R : \mathbf{X} \to \mathbf{Y}$	FD where $\mathbf{X}$ implies $\mathbf{Y}$ in R
Δ	set of FDs
$\langle n \log n, \log n \rangle$	direct access with $O(n \log n)$ preprocessing and $O(\log n)$ per access
$\langle 1, n \log n \rangle$	selection in $O(n \log n)$

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments and Florent Capelli for his input on simplifying the proof of Lemma 6.6.

REFERENCES

[1] Amir Abboud and Virginia Vassilevska Williams. 2014. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*. 434–443. DOI: https://doi.org/10.1109/FOCS.2014.53

[2] Nir Ailon and Bernard Chazelle. 2005. Lower bounds for linear degeneracy testing. *J. ACM* 52, 2 (2005), 157–171. DOI: https://doi.org/10.1145/1059513.1059515
[3] Noga Alon, Raphael Yuster, and Uri Zwick. 1997. Finding and counting given length cycles. *Algorithmica* 17, 3 (1997), 209–223. DOI: https://doi.org/10.1007/BF02523189
[4] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. 2007. On acyclic conjunctive queries and constant delay enumeration. In *CSL*. 208–222. DOI: https://doi.org/10.1007/978-3-540-74915-8_18
[5] Guillaume Bagan, Arnaud Durand, Etienne Grandjean, and Frédéric Olive. 2008. Computing the jth solution of a first-order query. *RAIRO-Theoret. Inform. Applic.* 42, 1 (2008), 147–164. DOI: https://doi.org/10.1051/ita:2007046
[6] Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. 2005. Subquadratic algorithms for 3SUM. In *Algorithms and Data Structures*. 409–421. DOI: https://doi.org/10.1007/11534273_36
[7] Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. 2020. Constant delay enumeration for conjunctive queries: A tutorial. *ACM SIGLOG News* 7, 1 (2020), 4–33. DOI: https://doi.org/10.1145/3385634.3385636
[8] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2017. Answering conjunctive queries under updates. In *PODS*. 303–318. DOI: https://doi.org/10.1145/3034786.3034789
[9] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. 1973. Time bounds for selection. *J. Comput. Syst. Sci.* 7, 4 (1973), 448–461. DOI: https://doi.org/10.1016/S0022-0000(73)80033-9
[10] Pierre Bourhis, Alejandro Grez, Louis Jachiet, and Cristian Riveros. 2021. Ranked enumeration of MSO logic on words. In *ICDT*. 20:1–20:19. DOI: https://doi.org/10.4230/LIPIcs.ICDT.2021.20
[11] Johann Brault-Baron. 2013. *De la Pertinence de l'énumération: Complexité en Logiques Propositionnelle et du Premier Ordre*. Ph.D. Dissertation. U. de Caen. Retrieved from: https://hal.archives-ouvertes.fr/tel-01081392.
[12] Nofar Carmeli and Markus Kröll. 2020. Enumeration complexity of conjunctive queries with functional dependencies. *Theor. Comput. Syst.* 64, 5 (2020), 828–860. DOI: https://doi.org/10.1007/s00224-019-09937-9
[13] Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. 2021. Tractable orders for direct access to ranked answers of conjunctive queries. In *PODS*. 325–341. DOI: https://doi.org/10.1145/3452021.3458331
[14] Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Benny Kimelfeld, and Nicole Schweikardt. 2020. Answering (unions of) conjunctive queries using random access and random-order enumeration. In *PODS*. 393–409. DOI: https://doi.org/10.1145/3375395.3387662
[15] Shaleen Deep and Paraschos Koutris. 2021. Ranked enumeration of conjunctive query results. In *ICDT*. 5:1–5:19. DOI: https://doi.org/10.4230/LIPIcs.ICDT.2021.5
[16] Arnaud Durand. 2020. Fine-grained complexity analysis of queries: From decision to counting and enumeration. In *PODS*. 331–346. DOI: https://doi.org/10.1145/3375395.3389130
[17] Jeff Erickson. 1995. Lower bounds for linear satisfiability problems. In *SODA*. 388–395. https://dl.acm.org/doi/10.5555/313651.313772.
[18] Robert W. Floyd and Ronald L. Rivest. 1975. Expected time bounds for selection. *Commun. ACM* 18, 3 (1975), 165–172. DOI: https://doi.org/10.1145/360680.360691
[19] Greg N. Frederickson. 1993. An optimal algorithm for selection in a min-heap. *Inf. Comput.* 104, 2 (1993), 197–214. DOI: https://doi.org/10.1006/inco.1993.1030
[20] Greg N. Frederickson and Donald B. Johnson. 1984. Generalized selection and ranking: Sorted matrices. *SIAM J. Comput.* 13, 1 (1984), 14–30. DOI: https://doi.org/10.1137/0213002
[21] Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. 2015. The complexity of resilience and responsibility for self-join-free conjunctive queries. *Proc. VLDB Endow.* 9, 3 (2015), 180–191. DOI: https://doi.org/10.14778/2850583.2850592
[22] Anka Gajentaan and Mark H. Overmars. 1995. On a class of O(n2) problems in computational geometry. *Computat. Geom.* 5, 3 (1995), 165–185. DOI: https://doi.org/10.1016/0925-7721(95)00022-2
[23] Wolfgang Gatterbauer and Dan Suciu. 2015. Approximate lifted inference with probabilistic databases. *Proc. VLDB Endow.* 8, 5 (2015), 629–640. DOI: https://doi.org/10.14778/2735479.2735494
[24] Wolfgang Gatterbauer and Dan Suciu. 2017. Dissociation and propagation for approximate lifted inference with standard relational database management systems. *VLDB J.* 26, 1 (2017), 5–30. DOI: https://doi.org/10.1007/s00778-016-0434-5
[25] Martin Charles Golumbic. 1980. *Algorithmic Graph Theory and Perfect Graphs*. Elsevier, 81–104. DOI: https://doi.org/10.1016/C2013-0-10739-8
[26] Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. 2016. Hypertree decompositions: Questions and answers. In *PODS*. 57–74. DOI: https://doi.org/10.1145/2902251.2902309
[27] Etienne Grandjean. 1996. Sorting, linear time and the satisfiability problem. *Ann. Math. Artif. Intell.* 16, 1 (1996), 183–236. DOI: https://doi.org/10.1007/BF02127798
[28] Egbert Harzheim. 2006. *Ordered Sets*. Vol. 7. Springer Science & Business Media. DOI: https://doi.org/10.1007/b104891

[29] Russell Impagliazzo and Ramamohan Paturi. 2001. On the complexity of K-SAT. *J. Comput. Syst. Sci.* 62, 2 (2001), 367–375. DOI: https://doi.org/10.1006/jcss.2000.1727
[30] Donald B. Johnson and Tetsuo Mizoguchi. 1978. Selecting the Kth element in $X + Y$ and $X_1 + X_2 + \cdots + X_m$. *SIAM J. Comput.* 7, 2 (1978), 147–153. DOI: https://doi.org/10.1137/0207013
[31] Jens Keppeler. 2020. *Answering Conjunctive Queries and FO+MOD Queries under Updates*. Ph.D. Dissertation. Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät. DOI: https://doi.org/10.18452/21483
[32] Benny Kimelfeld. 2012. A dichotomy in the complexity of deletion propagation with functional dependencies. In *PODS*. 191–202. DOI: https://doi.org/10.1145/2213556.2213584
[33] Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. 2018. Tight hardness for shortest cycles and paths in sparse graphs. In *SODA*. 1236–1252. DOI: https://doi.org/10.1137/1.9781611975031.80
[34] Stefan Mengel. 2021. A short note on the counting complexity of conjunctive queries. *CoRR* abs/2112.01108 (2021).
[35] A. Mirzaian and E. Arjomandi. 1985. Selection in X + Y and matrices with sorted rows and columns. *Inf. Process. Lett.* 20, 1 (1985), 13–17. DOI: https://doi.org/10.1016/0020-0190(85)90123-1
[36] Dan Olteanu and Maximilian Schleich. 2016. Factorized databases. *SIGMOD Rec.* 45, 2 (2016), 5–16. DOI: https://doi.org/10.1145/3003665.3003667
[37] Dan Olteanu and Jakub Zavodny. 2012. Factorised representations of query results: Size bounds and readability. In *ICDT*. 285–298. DOI: https://doi.org/10.1145/2274576.2274607
[38] Mihai Patrascu. 2010. Towards polynomial lower bounds for dynamic problems. In *STOC*. 603. DOI: https://doi.org/10.1145/1806689.1806772
[39] Mihai Pătraşcu and Ryan Williams. 2010. On the possibility of faster SAT algorithms. In *SODA*. 1065–1075. DOI: https://doi.org/10.1137/1.9781611973075.86
[40] Nikolaos Tziavelis, Deepak Ajwani, Wolfgang Gatterbauer, Mirek Riedewald, and Xiaofeng Yang. 2020. Optimal algorithms for ranked enumeration of answers to full conjunctive queries. *Proc. VLDB Endow.* 13, 9 (2020), 1582–1597. DOI: https://doi.org/10.14778/3397230.3397250
[41] Nikolaos Tziavelis, Wolfgang Gatterbauer, and Mirek Riedewald. 2020. Optimal join algorithms meet top-k. In *SIGMOD*. 2659–2665. DOI: https://doi.org/10.1145/3318464.3383132
[42] Nikolaos Tziavelis, Wolfgang Gatterbauer, and Mirek Riedewald. 2021. Beyond equi-joins: Ranking, enumeration and factorization. *Proc. VLDB Endow.* 14, 11 (2021), 2599–2612. DOI: https://doi.org/10.14778/3476249.3476306
[43] Nikolaos Tziavelis, Wolfgang Gatterbauer, and Mirek Riedewald. 2022. Any-k algorithms for enumerating ranked answers to conjunctive queries. *CoRR* abs/2205.05649 (2022).
[44] Virginia Vassilevska Williams. 2015. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *IPEC*. 17–29. DOI: https://doi.org/10.4230/LIPIcs.IPEC.2015.17
[45] Xiaofeng Yang, Mirek Riedewald, Rundong Li, and Wolfgang Gatterbauer. 2018. Any-k algorithms for exploratory analysis with conjunctive queries. In *ExploreDB*. 1–3. DOI: https://doi.org/doi.org/10.1145/3214708.3214711
[46] Mihalis Yannakakis. 1981. Algorithms for acyclic database schemes. In *VLDB*. 82–94. Retrieved from: https://dl.acm.org/doi/10.5555/1286831.1286840.
[47] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random sampling over joins revisited. In *SIGMOD*. 1525–1539. DOI: https://doi.org/10.1145/3183713.3183739

Received 7 December 2021; revised 7 December 2021; accepted 14 November 2022