Tight Fine-Grained Bounds for Direct Access on Join Queries

Karl Bringmann
Saarland University and Max Planck Institute for Informatics
Saarland Informatics Campus
Saarbrücken, Germany

Nofar Carmeli
DI ENS, ENS, CNRS, PSL University
Inria
Paris, France

Stefan Mengel
Univ. Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL)
Lens, France

ABSTRACT

We consider the task of lexicographic direct access to query answers. That is, we want to simulate an array containing the answers of a join query sorted in a lexicographic order chosen by the user. A recent dichotomy showed for which queries and orders this task can be done in polylogarithmic access time after quasilinear preprocessing, but this dichotomy does not tell us how much time is required in the cases classified as hard. We determine the preprocessing time needed to achieve polylogarithmic access time for all self-join free queries and all lexicographical orders. To this end, we propose a decomposition-based general algorithm for direct access on join queries. We then explore its optimality by proving lower bounds for the preprocessing time based on the hardness of a certain online Set-Disjointness problem, which shows that our algorithm's bounds are tight for all lexicographic orders on self-join free queries. Then, we prove the hardness of Set-Disjointness based on the Zero-Clique Conjecture which is an established conjecture from fine-grained complexity theory. We also show that similar techniques can be used to prove that, for enumerating answers to Loomis-Whitney joins, it is not possible to significantly improve upon trivially computing all answers at preprocessing. This, in turn, gives further evidence (based on the Zero-Clique Conjecture) to the enumeration hardness of self-join free cyclic joins with respect to linear preprocessing and constant delay.

CCS CONCEPTS

• **Theory of computation** → *Computational complexity and cryptography*; **Database query processing and optimization (theory)**; *Database query languages (principles).*

KEYWORDS

join queries, fine-grained complexity, direct access, enumeration

Karl Bringmann: This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979). *Nofar Carmeli and Stefan Mengel:* This work has been funded 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) and by the ANR project EQUUS ANR-19-CE48-0019.

PODS '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9260-0/22/06…$15.00
https://doi.org/10.1145/3517804.3526234

ACM Reference Format:
Karl Bringmann, Nofar Carmeli, and Stefan Mengel. 2022. Tight Fine-Grained Bounds for Direct Access on Join Queries. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3517804.3526234

1 INTRODUCTION

Algorithms for direct access allow to access answers to a query on a database essentially as if they are materialized and stored in an array: given an index $j \in \mathbb{N}$, the algorithm returns the jth answer (or an out-of-bounds error) in very little time. To make this possible, the algorithm first runs a preprocessing on the database that then allows answering arbitrary access queries efficiently. As the number of answers to a database query may be orders-of-magnitude larger than the size of the database, the goal is to avoid materializing all the answers during preprocessing and only simulate the array.

The direct access task (previously also called jth answer and random access) was introduced by Bagan et al. [7]. They also mentioned that this task can be used for uniform sampling (by first counting and drawing a random index) or for enumerating all query answers (by consecutively accessing all indices). They devised an algorithm that runs in only linear preprocessing time and average constant time per access call for a large class of queries (first-order queries) on databases of bounded degree. Direct access algorithm were also considered for queries in monadic second order logic on databases of bounded treewidth [6]. To reason about general databases and still expect extremely efficient algorithms, another approach is to restrict the class of queries instead. Follow-up work gave linear preprocessing and logarithmic access algorithms for subclasses of conjunctive queries over general databases [9, 13, 15]. There, it was also explained how direct access can be used to sample without repetitions [13].

Though all of the algorithms we mentioned above simulate storing the answers in a lexicographic order (whether they state it or not), one shortcoming they have in common is that the specific lexicographic order cannot be chosen by the user (but rather it depends on the query structure). Allowing the user to specify the order is desirable because then direct access can be used for additional tasks that are sensitive to the order, such as median finding and boxplot computation. Progress in this direction was recently made by Carmeli et al. [12] who identified which combinations of a conjunctive query and a lexicographic order can be accessed with linear preprocessing time and logarithmic access time. They identified an easy-to-check substructure of the query, called *disruptive trios*, whose (non-)existence distinguishes the tractable cases (w.r.t. the time guarantees we mentioned) from the intractable ones. In particular, if we consider acyclic join queries, they suggested an

algorithm that works for any lexicographic order that does not have disruptive trios with respect to the query. If the join query is also self-join free, they proved conditional lower bounds stating that for all other variable orders, direct access with polylogarithmic direct access requires superlinear time preprocessing. These hardness results assume the hardness of Boolean matrix multiplication. Given the known hardness of self-join free cyclic joins, if we also assume the hardness of hyperclique detection in hypergraphs, this gives a dichotomy for all self-join free join queries and lexicographic orders: they are tractable if and only if the query is acyclic with no disruptive trio with respect to the order.

What happens if the query and order we want to compute happen to fall on the intractable side of the dichotomy? This question was left open by previous work, and we aim to understand how much preprocessing is needed to achieve polylogarithmic access time for each combination of query and order. To this end, we introduce *disruption-free decompositions* of a query with respect to a variable order. These can be seen as hypertree decompositions of the queries, induced by the desired variable orders, that resolve incompatibilities between the order and the query. Practically, these decompositions specify which relations should be joined at preprocessing in order to achieve an equivalent acyclic join query with no disruptive trios. We can then run the known direct access algorithm from [12] with linear time preprocessing on the result of this preprocessing to get an algorithm for the query and order at hand with logarithmic access time. The cost of our preprocesing phase is therefore dominated by the time it takes to join the selected relations. We define the *incompatibility number* of a query and order and show that the preprocessing time of our solution is polynomial where the exponent is this number. Intuitively, the incompatibility number is 1 when the query is acyclic and the order is compatible, and it grows as the incompatibility between them grows.

Next, we aspire to know whether our solution can be improved. Though we can easily show that no other decomposition can be better than the specific decomposition we propose, we can still wonder whether a better algorithm can be achieved using an alternative technique. Thus, we set out to prove conditional lower bounds. Such lower bounds show hardness independently of a specific algorithmic technique, but instead they assume that some known problem cannot be solved significantly faster than by the state-of-the art algorithms. We show that the incompatibility number corresponds in a sense to the number of leaves of the largest star query that can be embedded in our given query. We then prove lower bounds for queries that allow embedding stars through a reduction from online k-Set-Disjointness. In this problem, we are given during preprocessing k sets of subsets of the universe, and then we need to answer queries that specify one subset from each set and ask whether these subsets are disjoint. On a technical level, in case the query is acyclic, the link of these hardness results with the existence of disruptive trios is that the latter correspond exactly to the possibility to embed a star with two leaves.

Using known hardness results for 2-Set-Disjointness, our reduction shows that the acyclic hard cases in the known dichotomy need at least quadratic preprocessing, unless both the 3-SUM Conjecture and the APSP Conjecture fail. These are both central, well-established conjectures in fine-grained complexity theory, see e.g. the survey [22]. To have tighter lower bounds for the case that the incompatibility number is not 2, we show the hardness of k-Set-Disjointness through a reduction from the *Zero-k-Clique Conjecture*. This conjecture postulates that deciding whether a given edge-weighted n-node graph contains a k-clique of total edge weight 0 has no algorithm running in time $O(n^{k-\varepsilon})$ for any $\varepsilon > 0$. For $k = 3$ this conjecture is implied by the 3SUM Conjecture and the APSP Conjecture, so it is very believable. For $k > 3$ the Zero-k-Clique Conjecture is a natural generalization of the case $k = 3$, and it was recently used in several contexts [1, 2, 4, 5, 11, 17]. Assuming the Zero-k-Clique Conjecture, we prove that the preprocessing time of our decomposition-based algorithm is (near-)optimal.

To conclude, our main result is as follows: a join query and an ordering of its variables with incompatibility number ι admit a lexicographic direct access algorithm with preprocessing time $O(|D|^{\iota})$ and logarithmic access time. Moreover, if the query is self-join free and the Zero-k-Clique conjecture holds for all k, there is no lexicographic direct access algorithm for this query and order with preprocessing time $O(|D|^{\iota-\epsilon})$ and polylogarithmic access time for any $\varepsilon > 0$.

As we develop our lower bound results, we notice that our techniques can also be used in the context of constant delay enumeration of query answers. We show that, assuming the Zero-k-Clique Conjecture, the preprocessing of any constant delay enumeration algorithm for the k-variable Loomis-Whitney join is roughly at least $O(|D|^{1+1/(k-1)})$ which tightly matches the trivial algorithm in which the answers are materialzed during preprocessing using a worst-case optimal join algorithm [19–21]. From the lower bound for Loomis-Whitney joins, we then infer the hardness of other cyclic joins using a construction by Brault-Baron [9]. Specifically, we conclude that the self-join free join queries that allow constant delay enumeration after linear processing are exactly the acyclic ones, unless the Zero-k-Clique Conjecture fails for some k. This dichotomy has been established based on the hardness of hyperclique detection in hypergraphs [9], see also [8], and we here give more evidence for this dichotomy by showing it also holds under a different complexity assumption.

2 PRELIMINARIES

2.1 Databases and Queries

A *join query* Q is an expression of the form $Q(\vec{u}) :- R_1(\vec{v}_1), \ldots, R_\ell(\vec{v}_n)$, where each R_i is a relation symbol, $\vec{v}_1, \ldots, \vec{v}_n$ are tuples of variables, and $\vec{u}$ is a tuple of all variables in $\bigcup_{i=1}^{n} \vec{v}_i$. Each $R_i(\vec{v}_i)$ is called an *atom* of Q. A query is called *self-join free* if no relation symbol appears in two different atoms. If the same relation symbol appears in two different atoms, $R_i = R_j$, then $\vec{v}_i$ and $\vec{v}_j$ must have the same arity. A *conjunctive query* is defined as a join query, with the exception that $\vec{u}$ may contain any subset of the query variables. We say that the query variables that do not appear in $\vec{u}$ are *projected*. Defined this way, join queries are simply conjunctive queries without projections.

The input to a query Q is a *database* D which assigns every relation symbol R_i in the query with a relation R_i^D: a finite set of tuples of constants, each tuple having the same arity as $\vec{v}_i$. The *size* $|D|$ of D is defined as the total number of tuples in all of its relations. An *answer* to a query Q over a database D is determined by a mapping μ from the variables of Q to the constants of D such

that $\mu(\vec{v}_i) \in R_i^D$ for every atom $R_i(\vec{v}_i)$ in Q. The answer is then $\mu(\vec{u})$. The set of all answers to Q over D is denoted $Q(D)$.

A *lexicographic order* of a join query Q is specified by a permutation L of the query variables. We assume databases come with an order on their constants. Then, L defines an order over $Q(D)$: the order between two answers is the same as the order between their assignments to the first variable in L on which their assignments differ.

We consider three types of tasks where the problem is defined by a query and the input is a database. *Testing* is the task where the user specifies a tuple of constants, and we need to determine whether this tuple is an answer to query over the database. *Enumeration* is the task of listing all query answers. We measure the efficiency of an enumeration algorithm with two parameters: the time before the first answer, which we call *preprocessing*, and the time between successive answers, which we call *delay*. Finally, *direct-access* in lexicographic order is a task defined by a query and an order where, after a preprocessing phase, the user can specify an index j and expect the jth answer in that order or an out-of-bounds error if there are less than j answers. We call the time it takes to provide an answer given an index the *access time*.

2.2 Hypergraphs

A hypergraph $H = (V, E)$ consist of a finite set V of vertices and a set E of edges, i.e., subsets of V. Given a set $S \subseteq V$, the hypergraph $H[S]$ induced by S is (S, E_S) with $E_S = \{e \cap S \mid e \in E\}$. A super-hypergraph H' of H is a hypergraph (V, E') such that $E \subseteq E'$. By $N_H(v)$ we denote the set of neighbors of a vertex v in H, i.e., $N_H(v) := \{u \in V \mid u \neq v, \exists e \in E : u, v \in e\}$. For $S \subseteq V$, we define $N_H(S) := \bigcup_{v \in S} N_H(v) \setminus S$. In these notations we sometimes leave out the subscript H when it is clear from the context.

A hypergraph H is called *acyclic* if we can eliminate all of its vertices by applying the following two rules iteratively:

- if there is an edge $e \in E$ that is completely contained in another edge $e' \in E$, delete e from H, and
- if there is a vertex $v \in V$ that is contained in a single edge $e \in E$, delete v from H, i.e., delete v from V and e.

An order in which the vertices can be eliminated in the above procedure is called an *elimination order* for H. Note that it might be possible to apply the above rules on several vertices or edges at the same moment. In that case, the order in which they are applied does not change the final result of the process.

A fractional edge cover of H is defined as a mapping $\mu : E \to [0, 1]$ such that for every $v \in V$ we have $\sum_{e: v \in e} \mu(e) \geq 1$. The weight of μ is defined as $\mu(E) := \sum_{e \in E} \mu(e)$. The fractional edge cover number is $\rho^*(H) := \min_\mu \mu(E)$ where the minimum is taken over all fractional edge covers of H. We remark that $\rho^*(H)$ and an optimal fractional edge cover of H can be computed efficiently by linear programming.

Every join query Q has an underlying hypergraph H, where the vertices of H correspond to the variables of Q and the edges of H correspond to the variable scopes of the atoms of Q. We use Q and H interchangably in our notation.

2.3 Known Algorithms

Here, we state some results that we will use in the remainder of this paper. All running time bounds are in the word-RAM model with $O(\log(n))$ bit words and unit-cost operations.

We use the following notions from [12]. Given a join query Q and a lexicographic order L, a *disruptive trio* consists of three variables x_1, x_2, x_3 such that x_3 appears after x_1 and x_2 in L, the variables x_1 and x_2 do not appear together in an atom of Q, but x_3 shares (different) atoms with both x_1 and x_2.

Theorem 1 ([12]). *If an acyclic join query Q and a lexicographic order L have no disruptive trios, then lexicographic direct access for Q and L can be solved with $O(|D|)$ preprocessing and $O(\log(|D|))$ access time.*

A celebrated result by Atserias, Marx and Grohe [3] shows that join queries of fractional edge cover number k, can, on any database D, result in query results of size at most $O(|D|^k)$, and this bound is tight for every query for some databases. The upper bound is made algorithmic by so-called worst-case optimal join algorithms.

Theorem 2 ([19–21]). *There is an algorithm that for every query Q of fractional edge cover number $\rho^*(Q)$, given a database D, computes $Q(D)$ in time $O(|D|^{\rho^*(Q)})$.*

2.4 Fine-Grained Complexity

Fine-grained complexity theory aims to find the exact exponent of the best possible algorithm for any problem, see e.g. [22] for a recent survey. Since unconditional lower bounds of this form are currently far out of reach, fine-grained complexity provides conditional lower bounds that hold when assuming a conjecture about some central, well-studied problem.

Some important reductions and algorithms in fine-grained complexity are randomized, i.e., algorithms are allowed to make random choices in their run and may return wrong results with a certain probability, see e.g. [18] for an introduction. Throughout the paper, when we write "randomized algorithm" we always mean a randomized algorithm with success probability at least 2/3. It is well-known that the success probability can be boosted to any $1-\delta$ by repeating the algorithm $O(\log(1/\delta))$ times and returning the majority result. In particular, we can assume to have success probability at least $1 - 1/n^{10}$, at the cost of only a factor $O(\log(n))$. Our reductions typically worsen the success probability by some amount, but using boosting it can be improved back to $1 - 1/n^{10}$; we do not make this explicit in our proofs.

We stress that randomization is only used in our hardness reductions, while all our algorithmic results are deterministic.

We will base our lower bounds on the following problem which is defined for every fixed constant $k \in \mathbb{N}, k \geq 3$.

Definition 3. In the Zero-k-Clique problem, given an n-node graph $G = (V, E)$ with edge-weights $w : E \to \{-n^c, \ldots, n^c\}$ for some constant c, the task is to decide whether there are vertices $v_1, \ldots, v_k \in V$ such that they form a k-clique (i.e. $\{v_i, v_j\} \in E$ for all $1 \leq i < j \leq k$) and their total edge-weight is 0 (i.e. $\sum_{1 \leq i < j \leq k} w(v_i, v_j) = 0$). In this case we say that $v_1, \ldots, v_k$ is a zero-clique.

We remark that by hashing techniques we can assume $c \leq 10k$.

The following conjectures have been used in several places, see e.g. [1, 2, 4, 5, 11, 17]. The first conjecture is for a fixed k, the second postulates hardness for all k.

Conjecture 1 (Zero-k-Clique Conjecture). *For no constant $\varepsilon > 0$ Zero-k-Clique has a randomized algorithm running in time $O(n^{k-\varepsilon})$.*

Conjecture 2 (Zero-Clique Conjecture). *For every $k \geq 3$ the Zero-k-Clique Conjecture is true.*

It is known that the Zero-3-Clique Conjecture, also called Zero-Triangle Conjecture, is implied by two other famous conjectures: the 3SUM Conjecture [23] and the APSP Conjecture [24]. Since we do not use these conjectures directly in this paper, we do not formulate them here and refer the interested reader to the survey [22].

We remark that instead of Zero-k-Clique some references work with the *Exact-Weight-k-*Clique problem, where we are additionally given a target weight t and want to find a k-clique of weight t. Both problems are known to have the same time complexity up to constant factors, see e.g. [1].

A related problem is *Min-k-*Clique, where we are looking for the k-clique of minimum weight. The Min-k-Clique Conjecture postulates that this problem also cannot be solved in time $O(n^{k-\varepsilon})$. It is known that the Min-k-Clique Conjecture implies the Zero-k-Clique Conjecture, see e.g. [1].

3 DIRECT-ACCESS ALGORITHM

In this section, we give an algorithm that, for every join query and desired lexicographic order, provides direct access to the query result on an input database. In particular, we propose to add new atoms to the query such that the resulting query has no disruptive trios with respect to the order. Then, any direct-access algorithm that assumes acyclicity and no disruptive-trios can be applied, provided we can compute a database for the new query that yields the same answers. In the full version of this paper [10], we show that the new query is essentially a generalized hypertree decomposition [14] of optimal fractional hypertree width out of all decompositions with the required properties. This shows that the suggested solution here is the best we can achieve using a decomposition, but it does not mean we cannot do better using a different method—the latter question is studied in the later sections of this paper.

3.1 Disruption-Free Decompositions

We describe a process that iteratively eliminates disruptive trios in a query by adding new atoms.

Definition 4 (Disruption-Free Decomposition)**.** Let $Q(\vec{V})$ be a join query and $L = (v_1, \ldots, v_\ell)$ an ordering of $\vec{V}$. Let H_ℓ be the hypergraph of Q, and for $i = \ell, \ldots, 1$ construct hypergraph H_{i-1} from H_i by adding an edge $e_i := \{v_i\} \cup \{v_j \mid j < i \text{ and } v_j \in N_{H_i}(v_i)\}$. The disruption-free decomposition of Q is then defined to be H_0.

Example 5. Consider the query $Q(x_1, x_2, x_3, x_4, x_5) :- R_1(x_1, x_5), R_2(x_2, x_4), R_3(x_3, x_4), R_4(x_3, x_5)$ with the order $(x_1, x_2, x_3, x_4, x_5)$ of its variables. Its graph is shown in Figure 1. In the first step of the construction of Definition 4, we add the edge $\{x_5\} \cup N(x_5) = \{x_1, x_3, x_5\}$. Similarly, in the second step, we add $\{x_2, x_3, x_4\}$. For the third step, note that $N(x_3) = \{x_1, x_2, x_4, x_5\}$ due to the edges

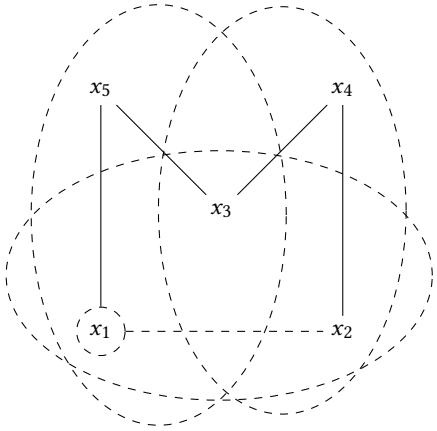

Figure 1: The hypergraph of Example 5. The original (hyper)graph of the query is drawn in full edges. The edges that we add in the construction are dashed.

we have added before. Out of these neighbors, only x_1 and x_2 come before x_3 in the order. So we add the edge $\{x_1, x_2, x_3\}$. Finally, for x_2, we add the edge $\{x_1, x_2\}$ and for x_1 the singleton edge $\{x_1\}$.

Proposition 6. *The disruption-free decomposition of a query Q is an acyclic super-hypergraph of the hypergraph of Q without any disruptive trios.*

It will be useful in the remainder of this section to have a non-iterative definition of disruption-free decompositions. Let S_i be the vertices in the connected component of v_i in $Q[v_i, \ldots, v_\ell]$. In particular, we have that $v_i \in S_i$.

Lemma 7. *The edges introduced in Definition 4 are $e_i = \{v_i\} \cup \{v_j \mid j < i \text{ and } v_j \in N_Q(S_i)\}$ for $i \in \{1, \ldots, \ell\}$.*

Example 8. Let us illustrate Lemma 7 with the query of Example 5: For the variable x_5 we have $S_5 = \{x_5\}$. Consequently, $e_5 = \{v_5\} \cup N_Q(x_5) = \{x_1, x_3, x_5\}$ which is exactly the edge that is added for x_5. The case for x_4 is analogous. For x_3 we have $S_3 = \{x_3, x_4, x_5\}$. Consequently $e_3 := \{x_3\} \cup N_Q(\{x_3, x_4, x_5\}) = \{x_1, x_2, x_3\}$. Finally, $S_2 = \{x_2, x_3, x_4, x_5\}$, so we add the edge $e_2 = \{x_1, x_2\}$. For x_1 we add again the singleton edge $\{x_1\}$.

3.2 The Algorithm

The idea of our direct access algorithm is to use the disruption-free decomposition. More precisely, we define a new query Q' from Q such that the hypergraph of Q' is the disruption-free decomposition of Q. Then, given an input database D for Q, we compute a new database D' for Q' such that $Q(D) = Q'(D')$. Since Q' has no disruptive trio, we can then use the algorithm from [12] on Q' and D' to allow direct access. A key component to making this approach efficient is the efficient computation of D'. To measure its complexity, we introduce the following notion.

Definition 9 (Incompatibility Number)**.** Let $Q(\vec{V})$ be a join query with hypergraph H and $L = (v_1, \ldots, v_\ell)$ be an ordering of $\vec{V}$. Let $\rho^*(H[e_i])$ be the fractional edge cover number of e_i as defined

in Definition 4. We call $\max_{i=1,\ldots,\ell} \rho^*(H[e_i])$ the *incompatibility number* of Q and L.

Note that we assume that queries have at least one atom, so the incompatibility number of any query and any order is at least 1. The incompatibility number can also be seen as the *fractional width* of the disruption-free decomposition, and we can show that this decomposition has the minimum fractional width out of all decompositions with the properties we need, see the full version of this paper for details [10]. We now show that the incompatibility number lets us state an upper bound for the computation of a database D' with the properties claimed above.

Theorem 10. *Given a join query and an ordering of its variables with incompatibility number ι, lexicographic direct access with respect to L can be achieved with $O(|D|^\iota)$ preprocessing and logarithmic access time.*

Proof. We construct Q' by adding for every edge e_i an atom R_i whose variables are those in e_i. We compute a relation $R_i^{D'}$ as follows: let μ be a fractional edge cover of $H[e_i]$ of weight ι. Let $\bar{e}_1, \ldots, \bar{e}_r$ be the edges of $H[e_i]$ that have positive weight in μ. For every $\bar{e}_j$ there is an edge e'_j of H such that $\bar{e}_j = e'_j \cap e_i$. Let $\bar{R}_j(\bar{X}_j)$ be the projection to e_i of the atom corresponding to e'_j. Let D^* be the extension of D that contains the corresponding projected relations for the $\bar{R}_j(\bar{X}_j)$. We set $R_i^{D'} := Q_i(D^*)$ where $Q_i(e_i) :- \bar{R}_1(X_1), \ldots, \bar{R}_r(X_r)$. Then clearly for all tuples $\vec{t}$ in $Q(D)$, the tuple we get by projecting $\vec{t}$ to e_i lies in $R_i^{D'}$. As a consequence, we can construct D' by adding all relations $R_i^{D'}$ to D to get $Q(D) = Q'(D')$. Moreover, the hypergraph of Q' is the disruption-free decomposition of the hypergraph of Q, so we can apply the algorithm from [12] for direct access.

It remains to show that all $R_i^{D'}$ can be constructed in time $O(|D|^\iota)$. To this end, consider the join query $Q_i(e_i)$ from before. By definition, its variable set is e_i, and μ is a fractional edge cover of weight at most ι. Thus, we can use a worst-case optimal join algorithm from Theorem 2 to compute $R_i^{D'}$ in time $O(|D|^\iota)$. □

4 SET-DISJOINTNESS-BASED HARDNESS FOR DIRECT ACCESS

In this section, we show lower bounds for lexicographically ranked direct access for all self-join free queries and all variable orders. We first reduce general direct access queries to the special case of star queries, which we introduce in Section 4.1. Then we show that lower bounds for direct access to star queries follow from lower bounds for k-Set-Disjointness, see Section 4.3. Later, in Section 5, we prove a lower bound for k-Set-Disjointness based on the Zero-Clique Conjecture.

4.1 From k-Star Queries to Direct Access

A crucial role in our reduction is played by the k-star query $Q_k^\star$:

$$Q_k^\star(x_1, \ldots, x_k, z) :- R_1(x_1, z), \ldots, R_k(x_k, z),$$

as well as its variant $\overline{Q}_k^\star$ in which variable z is projected away:

$$\overline{Q}_k^\star(x_1, \ldots, x_k) :- R_1(x_1, z), \ldots, R_k(x_k, z).$$

We will always denote their input database by $D^\star$. We say that a variable order L is *bad for* $Q_k^\star$ if z is the last variable in L.

Lemma 11. *Let $Q(\vec{V})$ be a self-join-free join query and let $L = (v_1, \ldots, v_\ell)$ be an ordering of $\vec{V}$. Let ι be the incompatibility number of Q and L and assume $\iota > 1$. If there is an $\varepsilon > 0$ such that for all $\delta > 0$ there is a direct access algorithm for Q and L with preprocessing time $O(|D|^{\iota-\varepsilon})$ and access time $O(|D|^\delta)$, then there is a $k \in \mathbb{N}$, $k > 2$ and $\varepsilon' > 0$ such that for all $\delta' > 0$ there is a direct access algorithm for $Q_k^\star$ with respect to a bad ordering with preprocessing time $O(|D^\star|^{k-\varepsilon'})$ and access time $O(|D^\star|^{\delta'})$.*

Proof. We will use the concept of fractional independent sets. A fractional independent set in a hypergraph $H = (V, E)$ is a mapping $\phi : V \to [0, 1]$ such that for every $e \in E$ we have $\sum_{v \in e} \phi(v) \leq 1$. The weight of ϕ is defined to be $\phi(V) = \sum_{v \in V} \phi(v)$. The fractional independent set number of H is defined as $\alpha^*(H) := \max_\phi \phi(V)$ where the maximum is taken over all fractional independent sets of H. Using linear programming duality, we have for every graph H and every vertex set S that $\alpha^*(H[S]) = \rho^*(H[S])$. We use the same notation on join queries, with the meaning of applying it to the underlying hypergraph.

We use the notation S_r and e_r as in Lemma 7. Let r be such that $\iota = \rho^*(H[e_r])$ where H is the hypergraph coresponding to Q. Then, by what we said above, we know that there is a fractional independent set $\phi : e_r \to [0, 1]$ such that $\sum_{v \in e_r} \phi(v) = \iota$. Since ϕ is the solution of a linear program with integer coefficients and weights, all values $\phi(v)$ are rational numbers. Let λ be the least common multiple of the denominators of $\{\phi(v) \mid v \in e_r\}$. Now define a new weight function $\phi' : e_r \to \{0, \ldots, \lambda\}$ by $\phi'(v) := \lambda\phi(v)$.

Let $k := \lambda\iota$. We will show how to simulate random access to $Q_k^\star$ by embedding it into Q. To this end, every $v \in e_r$ takes $\phi'(v)$ *roles*, that is we assign $\phi'(v)$ many of the variables $x_1, \ldots, x_k$ of $Q_k^\star$ to v. We do this in such a way that for all pairs $v, v' \in e_r$, whenever a variable v comes before v' in L, then for every role x_i of v and every role x_j of v' we have that $i < j$. Moreover, for every x_i we have that there is a unique variable $v \in e_r$ whose role is x_i. Note that $\sum_{v \in e_r} \phi'(v) = \lambda \sum_{v \in e_r} \phi_v = \lambda\iota = k$, so this distribution of roles is possible. Now add as an additional role the role z for every $v \in S_r$. Note that when doing so, the variable v_r can have both some roles x_i and the role z.

We now construct a database D for Q, given an input database $D^\star$ for $Q_k^\star$. For every variable x, denote by $\text{dom}^\star(x)$ the active domain in $D^\star$. We fix all variables of Q that have no role to a constant $\perp$. To simplify the reasoning, we will in the remainder ignore these variables, so only the variables in $e_r \cup S_r$ remain. Let us first sketch the embedding of $Q_k^\star$: the role of S_r is to encode the middle vertex z and connect it to all x_i. To this end, we encode the choice of a vertex z into each node in S_r. Since S_r induces a connected hypergraph, we can ensure that this choice of vertex z is consistent among all nodes in S_r. Since S_r has edges to all nodes in e_r, we can encode all relations $R_i(x_i, z)$. This allows us to simulate the star query $R_1(x_1, z), ..., R_k(x_k, z)$.

We now proceed with the technical details. For each variable $v \in e_r \cup S_r$ we define the domain as follows: let $y_1, \ldots, y_s$ be the roles of v given in the order defined by L. Then the domain of v is

$\mathrm{dom}(v) := \mathrm{dom}^\star(y_1) \times \ldots \times \mathrm{dom}^\star(y_s)$. We order $\mathrm{dom}(v)$ lexicographically. Intuitively, v takes the values of all its roles. To define the relations of D, consider two cases: if none of the variables in an atom $R(v_{i_1}, \ldots, v_{i_s})$ of Q plays the role z, the relation R^D is simply $\mathrm{dom}(v_{i_1}) \times \ldots \times \mathrm{dom}(v_{i_s})$. Otherwise, let $x_{j_1}, \ldots, x_{j_t}, z$ be the roles played by variables in $R(v_{i_1}, \ldots, v_{i_s})$. Then we compute the sub-join $J := Q_J(D^\star)$ where $Q_J(x_{j_1}, \ldots, x_{j_t}) \coloneq R_{j_1}(x_{j_1}, z), \ldots, R_{j_t}(x_{j_t}, z)$. Note that there is an injection $\nu : J \to \mathrm{dom}(v_{i_1}) \times \ldots \times \mathrm{dom}(v_{i_s})$ that consists of "packing" the values for the different variables into the variables of Q according to the roles in the following sense: for every tuple $\vec{t}$, ν maps each variable v_i with role set $\mathcal{R}$ to the tuple we get from $\vec{t}$ by deleting the coordinates not in $\mathcal{R}$. The relation R^D is then simply $\nu(J)$.

Note that the construction gives a bijection between $Q(D)$ and $Q_k^\star(D^\star)$: in all tuples t in $Q(D)$, all variables that have the role z must take the same value on the corresponding coordinate by construction, because S_r is connected. Moreover, for every atom $R_j(x_j, z)$ of $Q_k^\star$, there is an atom of Q containing (not necessarily different) variables v_x and v_z such that v_x has the role x_j and v_z has the role z. By construction it then follows that on the corresponding components the values v_x and v_z take values that are consistent with the relation $R_j^{D^\star}$. This directly gives the desired bijection. Also, given an answer to Q, we can easily compute the corresponding answer to $Q_k^\star$ by "unpacking", i.e., inverting ν. Hence, a direct access algorithm for Q gives a direct access algorithm for $Q_k^\star$. Moreover, by construction, the correct order is maintained by ν. It remains to analyze the running time of this algorithm. We first show that the constructed instance D can be computed reasonably efficiently.

Claim 1. *All relations in D can be computed in time $O(|D^\star|^\lambda)$.*

It follows that the size of D is at most $O(|D^\star|^\lambda)$. If for some $\varepsilon > 0$ and all $\delta > 0$ there is a direct access algorithm for Q and L with preprocessing time $O(|D|^{\iota-\varepsilon})$ and access time $O(|D|^\delta)$, then by running this algorithm on the constructed instance D we get direct access to $Q_k^\star$ with preprocessing time:

$$O(|D^\star|^\lambda + |D|^{\iota-\varepsilon}) = O(|D^\star|^\lambda + |D^\star|^{\lambda\iota-\lambda\varepsilon}) = O(|D^\star|^{k-\varepsilon'}),$$

where we used $\iota > 1$ and thus $\lambda < \lambda\iota - \varepsilon' = k - \varepsilon'$ for sufficiently small $\varepsilon' > 0$. For any desired $\delta' > 0$, by setting $\delta := \delta'/\lambda$ the access time of this algorithm is $O(|D|^\delta) = O(|D^\star|^{\lambda\delta}) = O(|D^\star|^{\delta'})$. This finishes the proof. □

The situation of Lemma 11 simplifies for acyclic queries.

Corollary 12. *Let $Q(\vec{V})$ be an acyclic self-join-free join query and L an order of $\vec{V}$ with incompatibility number $\iota > 1$. Then ι is an integer, and if Q has a direct access algorithm with preprocessing time $p(|D|)$ and access time $r(|D|)$, then so has $Q_\iota^\star$.*

4.2 From Projected Stars to Star Queries

Proposition 13. *Let L be a bad order on $x_1, \ldots, x_k, z$. If there is an algorithm that solves the direct access problem for $Q_k^\star(x_1, \ldots, x_k, z)$ and L with preprocessing time $p(|D|)$ and access time $r(|D|)$, then there is an algorithm for the testing problem for $\overline{Q_k^\star}(x_1, \ldots, x_k)$ with preprocessing time $p(|D|)$ and query time $O(r(|D|)\log(|D|))$.*

Proof. If an answer $(a_1, \ldots, a_k)$ is in $\overline{Q_k^\star}(D)$, then answers of the form $(a_1, \ldots, a_k, b)$ (for any b) form a contiguous sequence in $Q_k^\star(D)$, since the position of z has the lowest priority in L. Thus, a simple binary search using the direct access algorithm allows to test whether such a tuple $(a_1, \ldots, a_k, b) \in Q_k^\star(D)$ exists, using a logarithmic number of direct access calls. □

4.3 From Set-Disjointness to Projected Stars

We observe that the testing problem for the projected star query $\overline{Q_k^\star}$ is equivalent to the following problem:

Definition 14. In the k-Set-Disjointness problem, we are given an instance $\mathcal{I}$ consisting of a universe U and families $\mathcal{A}_1, \ldots, \mathcal{A}_k \subseteq 2^U$ of subsets of U. We denote the sets in family $\mathcal{A}_i$ by $S_{i,1}, \ldots, S_{i,|\mathcal{A}_i|}$. The task is to preprocess $\mathcal{I}$ into a data structure that can answer queries of the following form: Given indices $j_1, \ldots, j_k$, decide whether $S_{1,j_1} \cap \ldots \cap S_{k,j_k} = \emptyset$.

We denote the number of sets by $n := \sum_i |\mathcal{A}_i|$ and the input size by $\|\mathcal{I}\| := \sum_i \sum_{S \in \mathcal{A}_i} |S|$. We call $|U|$ the universe size.

Lemma 15. *If there is a testing algorithm for $\overline{Q_k^\star}$ with preprocessing time $p(|D|)$ and access time $r(|D|)$, then k-Set-Disjointness has an algorithm with preprocessing time $p(\|\mathcal{I}\|)$ and query time $r(\|\mathcal{I}\|)$.*

Proof. For each family $\mathcal{A}_i$, consider the relation

$$R_i^D := \{(j, v) \mid v \in S_{i,j}\}.$$

Let D be the resulting database, and note that $|D| = \|\mathcal{I}\|$. Then for every query $(j_1, \ldots, j_k)$ of the k-Set-Disjointness problem, we have $S_{1,j_1} \cap \ldots \cap S_{k,j_k} \neq \emptyset$ if and only if $(j_1, \ldots, j_k) \in \overline{Q_k^\star}(D)$. □

5 HARDNESS OF SET DISJOINTNESS

In this section, we will show lower bounds for k-Set Disjointness, as defined in Definition 14. In combination with the reductions from the previous section, this will give us lower bounds for direct access in Section 6. The main result of this section is the following.

Theorem 16. *If there is $k \in \mathbb{N}, k \geq 2$ and $\varepsilon > 0$ such that for all $\delta > 0$ there is an algorithm for k-Set-Disjointness that, given an instance $\mathcal{I}$, answers queries in time $O(\|\mathcal{I}\|^\delta)$ after preprocessing time $O(\|\mathcal{I}\|^{k-\varepsilon})$, then the Zero-$(k+1)$-Clique Conjecture is false.*

The case $k = 2$ follows from the literature [16, 25] as discussed in more detail in the full version of this paper [10]. In the following, we generalize these results to any $k \geq 2$. Our chain of reductions closely follows the proof of the case $k = 2$ by Vassilevska Williams and Xu [25], but also uses additional ideas required for the generalization to larger k.

5.1 k-Set-Intersection

We start by proving a lower bound for the following problem of listing many elements in a set intersection.

Definition 17. In the k-Set-Intersection problem, we are given an instance $\mathcal{I}$ consisting of a universe U and families $\mathcal{A}_1, \ldots, \mathcal{A}_k \subseteq 2^U$ of subsets of U. We denote the sets in family $\mathcal{A}_i$ by $S_{i,1}, \ldots, S_{i,|\mathcal{A}_i|}$. The task is to preprocess $\mathcal{I}$ into a data structure that can answer queries of the following form: Given indices $j_1, \ldots, j_k$ and a number

T, compute the set $S_{1,j_1} \cap \ldots \cap S_{k,j_k}$ if it has size at most T; if it has size more than T, then compute any T elements of this set.

We denote the number of sets by $n := \sum_i |\mathcal{A}_i|$ and the input size by $\|\mathcal{I}\| := \sum_i \sum_{S \in \mathcal{A}_i} |S|$. We call $|U|$ the universe size.

The main result of this section shows that k-Set-Intersection is hard in the following sense. Later we will use this result to show hardness for k-Set-Disjointness.

Theorem 18. *Assuming the Zero-$(k+1)$-Clique Conjecture, for every constant $\varepsilon > 0$ there exists a constant $\delta > 0$ such that no randomized algorithm solves k-Set-Intersection on universe size $|U| = n$ and $T = \Theta(n^{1-\varepsilon/2})$ in preprocessing time $O(n^{k+1-\varepsilon})$ and query time $O(Tn^{\delta})$.*

We prove Theorem 18 in the rest of this section.

Preparations. To start our reductions, first note that finding a zero-k-clique in a general graph is equivalent to finding a zero-k-clique in a complete k-partite graph. This can be shown using a reduction that duplicates the nodes and the edges k times, and replaces non-existing edges by edges of very large weight.

Observation 19 ([1]). *If the Zero-$(k+1)$-Clique Conjecture is true, then it is also true restricted to complete $(k+1)$-partite graphs.*

Due to this observation, we can assume that we are given a complete $(k+1)$-partite graph $G = (V, E)$ with n vertices and with color classes $V_1, \ldots, V_{k+1}$ and a weight function w on the edges. We denote by $w(u, v)$ the weight of the edge from u to v, and more generally by $w(v_1, \ldots, v_\ell) := \sum_{1 \le i < j \le \ell} w(v_i, v_j)$ the total weight of the clique $(v_1, \ldots, v_\ell)$.

For our construction it will be convenient to assume that the edge weights lie in a finite field. Recall that in the Zero-$(k+1)$-Clique problem we can assume that all weights are integers between $-n^c$ and n^c, for some constant c that may depend on k. We first compute a prime number between $10(k+1)^2 n^c$ and $100(k+1)^2 n^c$ by a standard algorithm: Pick a random number in that interval, check whether it is a prime, and repeat if it is not. By the prime number theorem, a random number in that interval is a prime with probability $\Theta(1/\log(n))$. It follows that in expected time $O(\text{polylog}(n))$ we find a prime p in that interval.

Having found the large prime p, we consider all edge weights as given in the finite field $\mathbb{F}_p$. Note that p is bigger than the sum of weights of any $(k+1)$-clique in G, so the $(k+1)$-cliques of weight 0 are the same over $\mathbb{Z}$ and $\mathbb{F}_p$. Thus, in the remainder we assume all arithmetic to be done over $\mathbb{F}_p$.

We now want to spread the edge weights evenly over $\mathbb{F}_p$. To this end, define a new weight function $w' : E \to \mathbb{F}_p$ by choosing independently and uniformly at random from $\mathbb{F}_p$:

- one value x,
- for all $v \in V_{k+1}$ and all $j \in [k-1]$ a value y_v^j.

We then define a new weight function w' by setting for any $i, j \in [k+1]$ and any $v \in V_i$ and $u \in V_j$:

$$w'(v,u) = x \cdot w(v,u) + \begin{cases} y_u^1, & \text{if } i = 1,\ j = k+1 \\ y_u^i - y_u^{i-1}, & \text{if } 2 \le i < k,\ j = k+1 \\ -y_u^{k-1}, & \text{if } i = k,\ j = k+1 \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Note that in every $(k+1)$-clique $(v_1, \ldots, v_{k+1}) \in V_1 \times \ldots \times V_{k+1}$ the sum $w'(v_1, \ldots, v_{k+1})$ contains every term $y_{v_{k+1}}^i$ for $i \in [k-1]$ once positively and once negatively, so for the overall weight we have $w'(v_1, \ldots, v_{k+1}) = x \cdot w(v_1, \ldots, v_{k+1})$. In particular, assuming $x \neq 0$, the zero-cliques remain the same and thus we can continue the construction with the weight function w'.

We next split $\mathbb{F}_p$ into n^ρ intervals of size $\lceil p/n^\rho \rceil$ or $\lfloor p/n^\rho \rfloor$, where ρ is a constant that we will choose later. In the following, I_i always denotes an interval resulting from this splitting of $\mathbb{F}_p$. We denote by S the set of all tuples $(I_0, \ldots, I_k)$ of intervals in $\mathbb{F}_p$ such that $0 \in \sum_{i=0}^k I_i$. Note that there are only $O(n^{\rho k})$ tuples in S and they can be enumerated efficiently: given $I_0, \ldots, I_{k-1}$, the set $-\sum_{i=0}^{k-1} I_i$ can be covered by $O(k) = O(1)$ intervals I_k, and these intervals can be computed efficiently.

For any clique $C = (v_1, \ldots, v_{k+1}) \in V_1 \times \ldots \times V_{k+1}$ there is a tuple $(I_0, \ldots, I_k)$ of intervals such that

$$\forall i \in [k] \colon w'(v_i, v_{k+1}) \in I_i \ \text{ and } \ w'(v_1, \ldots, v_k) \in I_0. \quad (2)$$

We call $(I_0, \ldots, I_k)$ the *weight interval tuple* of C. If C is a zero-clique, then its weight interval tuple must be in S.

The Reduction. With these preparations, we are now ready to present our reduction. For every weight interval tuple $(I_0, \ldots, I_k) \in S$ we construct an instance of k-Set-Intersection as follows. For any $i \in [k]$ we construct

$$\mathcal{A}_i := \{S_v \mid v \in V_i\} \ \text{ where } \ S_v := \{u \in V_{k+1} \mid w'(v,u) \in I_i\}.$$

Moreover, we consider the queries

$$Q := \{(v_1, \ldots, v_k) \in V_1 \times \ldots \times V_k \mid w'(v_1, \ldots, v_k) \in I_0\}.$$

We first run the preprocessing of k-Set-Intersection on the instance $\mathcal{I} = (\mathcal{A}_1, \ldots, \mathcal{A}_k)$ and then query each $q \in Q$ with parameter $T := \lceil 100 \cdot 3^k n^{1-k\rho} \rceil$. For each answer v_{k+1} returned by some query $(v_1, \ldots, v_k)$ we check whether $(v_1, \ldots, v_{k+1})$ is a zero-clique, and if it is then we return this zero-clique.

After repeating this construction for every weight interval tuple $(I_0, \ldots, I_k) \in S$, if we never found a zero-clique, then we return 'no'.

Correctness. Let us show correctness of the reduction. Since we explicitly test for each query answer whether it is a zero-clique, if G contains no zero-clique then our reduction returns 'no'.

We next show that if there exists a zero-clique then our reduction returns a zero-clique with probability at least .99. To this end, we use the following claim.

Claim 2. *Let $(v_1, \ldots, v_{k+1}) \in V_1 \times \ldots \times V_{k+1}$ be a zero-clique. Then, with probability at least .99, there are less than $100 \cdot 3^k n^{1-k\rho}$ vertices $u \in V_{k+1}$ such that $(v_1, \ldots, v_k, u)$ has the same weight interval tuple as $(v_1, \ldots, v_{k+1})$ and $(v_1, \ldots, v_k, u)$ is not a zero-clique.*

We defer the proof of this Claim 2 to the full version [10]. We here only show how the correctness proof can be shown with it. Fix any zero-clique $(v_1, \ldots, v_{k+1})$, and denote its weight interval tuple by $(I_0, \ldots, I_k)$. Note that in the instance constructed for $(I_0, \ldots, I_k)$ we query $(v_1, \ldots, v_k)$. With probability at least 0.99, this query has less than $100 \cdot 3^k n^{1-k\rho}$ "false positives", that is, vertices $u \in V_{k+1}$ such that $(v_1, \ldots, v_k, u)$ has weight interval tuple $(I_0, \ldots, I_k)$ and $(v_1, \ldots, v_k, u)$ is no zero-clique. Then by listing $T = \lceil 100 \cdot 3^k n^{1-k\rho} \rceil$ answers for the query $(v_1, \ldots, v_k)$, at least one answer

v must correspond to a zero-clique. (That is, we not necessarily find v_{k+1}, but we find some vertex v forming a zero-clique together with $v_1, \ldots, v_k$.) Hence, with probability at least .99 we find a zero-clique, if there exists one. This probability can be boosted to high probability (that is, probability $1 - 1/n^c$ for any desirable constant c) by repeating the reduction algorithm $O(\log n)$ times and returning a zero-clique if at least one repetition finds a zero-clique.

Running Time. It remains to analyze the running time and to set the parameter ρ. To this end, assume that for some $\varepsilon > 0$ k-Set-Intersection with $|U| = n$ and $T = \Theta(n^{1-\varepsilon/2})$ can be solved with preprocessing time $O(n^{k+1-\varepsilon})$ and query time $O(Tn^{\delta})$ for $\delta := \frac{\varepsilon}{4k}$, as promised by the theorem statement. Note that our constructed instances have universe size $|U| = |V_{k+1}| \leq n$, and $T = O(n^{1-k\rho})$, so by setting $\rho := \frac{\varepsilon}{2k}$ the k-Set-Intersection algorithm can be applied.

Over $O(n^{k\rho})$ instances (one for every $(I_0, \ldots, I_k) \in S$), the total preprocessing time is $O(n^{k+1+k\rho-\varepsilon}) = O(n^{k+1-\varepsilon/2})$, since $\rho = \frac{\varepsilon}{2k}$.

Let us count the total number of queries. Note that $(v_1, \ldots, v_k)$ is queried in the instance for $(I_0, \ldots, I_k) \in S$ iff $w'(v_1, \ldots, v_k) \in I_0$. Recall that for every $I_0, \ldots, I_{k-1}$ there are $O(1)$ choices for I_k such that $(I_0, \ldots, I_k) \in S$. Hence, each $(v_1, \ldots, v_k)$ is queried in $O(n^{(k-1)\rho})$ instances, as there are $O(n^{\rho})$ choices for each of $I_2, \ldots, I_{k-1}$ and $O(1)$ choices for I_k. The total number of queries is thus $O(n^{k+(k-1)\rho})$. Each query runs in time $O(Tn^{\delta}) = O(n^{1-k\rho+\delta})$, so the total query time over all our constructed instances is $O(n^{k+(k-1)\rho} \cdot n^{1-k\rho+\delta}) = O(n^{k+1+\delta-\rho}) = O(n^{k+1-\rho/2})$ since we set $\delta = \rho/2$.

In total, we obtain running time $O(n^{k+1-\rho/2})$ for Zero-$(k+1)$-Clique, contradicting the Zero-$(k+1)$-Clique Conjecture. This finishes the proof of Theorem 18.

5.2 Unique-k-Set-Intersection

As the next step, we show hardness of k-Set-Intersection for $T = 1$, and even if we get the promise that there is a unique answer to a query. We formally define this problem as follows.

Definition 20. The Unique-k-Set-Intersection problem has the same input $\mathcal{I} = (\mathcal{A}_1, \ldots, \mathcal{A}_k)$ with $\mathcal{A}_i = \{S_{i,1}, \ldots, S_{i,|\mathcal{A}_i|}\} \subseteq 2^U$ as k-Set-Intersection (cf. Definition 17). In the query we are given indices $j_1, \ldots, j_k$ and if the set $S_{1,j_1} \cap \ldots \cap S_{k,j_k}$ consists of exactly one element then we want to return that element; otherwise we can return the error element $\perp$ (or an arbitrary answer).

Again we write $n = \sum_i |\mathcal{A}_i|$. We show how to reduce Unique-k-Set-Intersection to k-Set-Intersection.

Lemma 21. *For every $0 < \theta \leq 1$ and every $\varepsilon > 0$, if there is a randomized algorithm for Unique-k-Set-Intersection on universe size $O(n^{\theta})$ with preprocessing time $\widetilde{O}(n^{k-\varepsilon})$ and query time $\widetilde{O}(n^{\delta})$, then there is a randomized algorithm for k-Set-Intersection on universe size $|U| = n$ and $T = \Theta(n^{1-\theta})$ with preprocessing time $\widetilde{O}(Tn^{k-\varepsilon})$ and query time $\widetilde{O}(Tn^{\delta})$.*

5.3 k-Set-Disjointness

Recall that the k-Set-Disjointness problem is defined similarly as k-Set-Intersection, except that a query should just decide whether the intersection $S_{1,j_1} \cap \ldots \cap S_{k,j_k}$ is empty or not. Here an instance $\mathcal{I}$ is given by a universe set U and families $\mathcal{A}_1, \ldots, \mathcal{A}_k \subseteq 2^U$, where we write $\mathcal{A}_i = \{S_{i,1}, \ldots, S_{i,|\mathcal{A}_i|}\}$. We again let $n = \sum_i |\mathcal{A}_i|$ and $\|\mathcal{I}\| = \sum_i \sum_{S \in \mathcal{A}_i} |S|$. We again measure preprocessing time and query time independently.

Lemma 22. *Let $0 < \theta < 1, \varepsilon, \delta > 0$ be constants. If there is a randomized algorithm for k-Set-Disjointness on universe size $|U| = \Theta(n^{\theta})$ with preprocessing $\widetilde{O}(n^{k-\varepsilon})$ and query time $\widetilde{O}(n^{\delta})$, then there is a randomized algorithm for Unique-k-Set-Intersection on universe size $|U| = \Theta(n^{\theta})$ with preprocessing time $\widetilde{O}(n^{k-\varepsilon})$ and query time $\widetilde{O}(n^{\delta})$.*

So far we measured running times in terms of n, the number of sets. Next we show how to obtain lower bounds in terms of the input size $\|\mathcal{I}\| = \sum_i \sum_{S \in \mathcal{A}_i} |S|$.

Lemma 23. *For any constants $\varepsilon, \delta > 0$, if there is an algorithm that solves k-Set-Disjointness with preprocessing time $\widetilde{O}(\|I\|^{k-\varepsilon})$ and query time $\widetilde{O}(\|\mathcal{I}\|^{\delta})$, then for any constant $0 < \theta \leq \frac{\varepsilon}{2k}$ there is an algorithm that solves k-Set-Disjointness on universe size $|U| = \Theta(n^{\theta})$ with preprocessing time $\widetilde{O}(n^{k-\varepsilon/2})$ and query time $\widetilde{O}(n^{\delta(1+\theta)})$.*

Proof. For every instance $\mathcal{I}$ of k-Set-Disjointness on universe size $|U| = \Theta(n^{\theta})$, we have that $\|\mathcal{I}\| = O(n^{1+\theta})$. Now on $\mathcal{I}$, the assumed algorithm for k-Set-Disjointness takes preprocessing time

$$\widetilde{O}(\|I\|^{k-\varepsilon}) = \widetilde{O}(n^{k+k\theta-\varepsilon-\theta\varepsilon}) = \widetilde{O}(n^{k-\varepsilon/2}),$$

where we used $\theta \leq \frac{\varepsilon}{2k}$ in the last step. Observing that the query time is $\widetilde{O}(\|\mathcal{I}\|^{\delta}) = \widetilde{O}(n^{\delta(1+\theta)})$ completes the proof. □

Chaining the reductions from the previous sections, it is now not too hard to show Theorem 16; see the full version [10] for details.

6 TIGHT BOUNDS FOR DIRECT ACCESS

In this section, we formulate and prove the main result of this paper on direct access, showing that the incompatibility number determines the required preprocessing time.

Theorem 24. *Let Q be a self-join free join query, L be an ordering of its variables, and ι be the incompatibility number of Q, L. Then there is an algorithm that allows lexicographic direct access with respect to the order induced by L with preprocessing time $O(|D|^{\iota})$ and logarithmic access time.*

Moreover, if there is a constant $\varepsilon > 0$ such that for all $\delta > 0$ there is an algorithm allowing lexicographic direct access with respect to the order induced by L with preprocessing time $O(|D|^{\iota-\varepsilon})$ and access time $O(|D|^{\delta})$, then the Zero-Clique Conjecture is false.

Proof sketch. The first part was shown in Theorem 10. The second part for $\iota > 1$ follows by chaining Lemma 11, Proposition 13, Lemma 15 and Theorem 16. For $\iota = 1$, it is not hard to show that with preprocessing $O(|D|^{1-\varepsilon})$ an algorithm cannot get sufficient information to answer correctly. □

Note that for every $\delta, \delta' > 0$ we have $O(\log(|D|)^{\delta}) \leq O(|D|^{\delta'})$. Therefore, if for some $\varepsilon, \delta > 0$ there was an algorithm with preprocessing time $O(|D|^{\iota-\varepsilon})$ and access time $O(\log(|D|)^{\delta})$, then for every $\delta' > 0$ there would also be an algorithm with preprocessing time $O(|D|^{\iota-\varepsilon})$ and access time $O(|D|^{\delta'})$. It follows that the lower bound in Theorem 24 implies the lower bound for polylogarithmic query time stated in the introduction.

We remark that from Theorem 24 we can also get a lower bound for direct access for unconstrained lexicographic order based on fractional hypertree width [14]; for details see the full version [10].

7 LOWER BOUNDS FOR ENUMERATION OF CYCLIC JOINS

In this section we sketch another application of the Zero-k-Clique Conjecture. Specifically, we consider enumeration lower bounds for cyclic joins, assuming the hardness of Zero-k-Clique. These lower bounds reprove a dichotomy from [9] under a different complexity assumption. Due to space constraints, we only give the outline of the proof here and refer to the full version [10] for all details.

To prove our lower bound, we consider Loomis-Whitney joins which are queries of the form

$$LW_k(x_1, \ldots, x_k) \coloneq R_1(\bar{X}_1), \ldots, R_k(\bar{X}_k)$$

where for each $i \in [k]$, we have that $\bar{X}_i = x_1, \ldots x_{i-1}, x_{i+1} \ldots, x_k$. When $k = 3$, LW_3 is simply the (colored) triangle query:

$$LW_3(x_1, x_2, x_3) \coloneq R_1(x_2, x_3), R_2(x_1, x_3), R_3(x_1, x_2)$$

Loomis-Whitney joins are thus generalizations of the triangle join where instead of 3 nodes in a graph where every 2 share an edge, we are looking for k nodes in a hypergraph where every $k-1$ nodes share an edge. Loomis-Whitney joins are of special interest because, as we discuss below, they are in a well-defined sense the obstructions to a query being acyclic.

There is a naive way of designing constant delay algorithms for Loomis-Whitney joins: one can materialize the query result during preprocessing and then simply read them from memory in the enumeration phase. This solution requires preprocessing time $O(|D|^{1+1/(k-1)})$ using the algorithm by Ngo et al. [20] for Loomis-Whitney joins or, since LW_k has fractional cover size $1 + 1/(k-1)$, any worst-case optimal join algorithm from Theorem 2 will give the same guarantees. The following theorem shows that this naive approach is likely optimal.

Theorem 25. *For every $k \geq 3$ and every $\varepsilon > 0$ there is a $\delta > 0$ such that there is no enumeration algorithm for LW_k with preprocessing time $O(|D|^{1+\frac{1}{k-1}-\varepsilon})$ and delay $O(|D|^{\delta})$, assuming the Zero-k-Clique Conjecture.*

Theorem 25 is shown by a reduction from a version of set-intersection in which, instead of querying combinations of sets, one is supposed to enumerate all non-empty set-intersections of a given instance. Techniques similar to that in Section 5 can be used to show lower bounds for this enumeration variant of set-intersection based on the hardness of Zero-k-Clique. A reduction to enumeration for Loomis-Whitney joins then finishes the proof of Theorem 25; see [10] for all details.

We now describe the implication of hardness of Loomis-Whitney joins on the hardness of other cyclic joins, and derive an enumeration lower bound for self-join free cyclic joins based of Zero-k-Clique. A known characterizarion of cyclic hypergraphs is that they contain a chordless cycle or a non-conformal clique (i.e., a set of pairwise neighbors that are not contained in an edge). By considering a minimal non-conformal clique, Brault-Baron [9] used this property to claim that every cyclic query contains either a chordless cycle or a set of k variables such that there is no atom that contains all of them, but for every subset of size $k-1$ there is such an atom. In other words, by removing all occurrences of some variables, and removing some atoms whose variables are contained in other atoms, we can obtain either a Loomis-Whitney join or a chordless cycle join. In case of a chordless cycle, we can always use it to solve a chordless cycle of length 3 (also called a triangle), which can be seen as a Loomis-Whitney join of size $k = 3$.

An *exact reduction* from an enumeration problem P_1 to an enumeration problem P_2 constructs an input I_2 to P_2 given an input I_1 to P_1 in linear time such that there is a bijection that can be computed in constant time from the answers to P_2 over I_2 to the answers of P_1 over I_1. Note that enumeration in linear preprocessing time and constant delay is closed under exact reductions.

Stated in different words, Brault-Baron [9] essentially proved the following lemma:

Lemma 26. *Let Q be a self-join free cyclic join. There exists $k \geq 3$ such that there is an exact reduction from LW_k to Q.*

By combining Lemma 26 with Theorem 25, we conclude the enumeration hardness of cyclic joins based on the hardness of Zero-k-Clique: they have no enumeration algorithm with linear preprocessing time and constant delay.

Theorem 27. *Let Q be a self-join free cyclic join. Assuming the Zero-Clique Conjecture, there exists an $\varepsilon > 0$ such that there is no enumeration algorithm for Q with preprocessing time $O(|D|^{1+\varepsilon})$ and delay $O(|D|^{\varepsilon})$.*

8 CONCLUSION

In this paper, we identified a parameter that we call the incompatibility number of a join query and an order of its variables. We proposed an algorithm for direct access to the answers of a join query in the lexicographic order induced by the variable order, using a reduction to the acyclic case without disruptive trios which had been studied in [12]. The exponent of the preprocessing phase of this algorithm is exactly the incompatibility number. We then complemented this upper bound by a matching lower bound that shows that the incompatibility number cannot be beaten as the exponent of the preprocessing in the case of self-join free join queries, assuming the Zero-Clique Conjecture from fine-grained complexity theory. With only a minor change to the proof of our direct-access lower bounds, we also showed lower bounds for enumeration. First, we showed that for Loomis-Whitney joins we cannot significantly improve upon a simple algorithm that computes all answers during preprocessing, assuming the Zero-Clique Conjecture. Then, under the same assumption, we showed that acyclic queries are the only self-join free join queries whose answers can be enumerated in constant delay after linear preprocessing. This gives further evidence to a dichotomy that was already shown using a different complexity hypothesis.

Let us close with several directions for future research. First, we feel that it would be interesting to understand the role of self-joins for direct access. Are there queries with self-joins where the runtime is better than that of our algorithm? Or can our lower bounds be shown also in this setting under reasonable assumptions? Similarly, what can be said about queries with projections? Finally, if we only require the answers to be ordered by some of the attributes, while

the order of the other attributes is arbitrary, how does this change the complexity of direct access? On a more technical level, we have seen that the Zero-Clique Conjecture is tightly linked to k-Set-Intersection and k-Set-Disjointness, which can be seen as variants of a join query. Thus, it seems plausible that there are additional applications of the Zero-Clique Conjecture in database theory, and it would be interesting to further explore such connections.

REFERENCES

[1] Amir Abboud, Karl Bringmann, Holger Dell, and Jesper Nederlof. 2018. More consequences of falsifying SETH and the orthogonal vectors conjecture. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, Ilias Diakonikolas, David Kempe, and Monika Henzinger (Eds.). ACM, 253–266. https://doi.org/10.1145/3188745.3188938

[2] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. 2014. Consequences of Faster Alignment of Sequences. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 8572)*, Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias (Eds.). Springer, 39–51. https://doi.org/10.1007/978-3-662-43948-7_4

[3] Albert Atserias, Martin Grohe, and Dániel Marx. 2013. Size Bounds and Query Plans for Relational Joins. *SIAM J. Comput.* 42, 4 (2013), 1737–1767. https://doi.org/10.1137/110859440

[4] Arturs Backurs, Nishanth Dikkala, and Christos Tzamos. 2016. Tight Hardness Results for Maximum Weight Rectangles. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy (LIPIcs, Vol. 55)*, Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 81:1–81:13. https://doi.org/10.4230/LIPIcs.ICALP.2016.81

[5] Arturs Backurs and Christos Tzamos. 2017. Improving Viterbi is Hard: Better Runtimes Imply Faster Clique Algorithms. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 311–321. http://proceedings.mlr.press/v70/backurs17a.html

[6] Guillaume Bagan. 2009. *Algorithmes et complexité des problèmes d'énumération pour l'évaluation de requêtes logiques. (Algorithms and complexity of enumeration problems for the evaluation of logical queries).* Ph. D. Dissertation. University of Caen Normandy, France. https://tel.archives-ouvertes.fr/tel-00424232

[7] Guillaume Bagan, Arnaud Durand, Etienne Grandjean, and Frédéric Olive. 2008. Computing the jth solution of a first-order query. *RAIRO Theor. Informatics Appl.* 42, 1 (2008), 147–164. https://doi.org/10.1051/ita:2007046

[8] Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. 2020. Constant delay enumeration for conjunctive queries: a tutorial. *ACM SIGLOG News* 7, 1 (2020), 4–33.

[9] Johann Brault-Baron. 2013. *De la pertinence de l'énumération: complexité en logiques propositionnelle et du premier ordre.* Ph. D. Dissertation. Université de Caen.

[10] Karl Bringmann, Nofar Carmeli, and Stefan Mengel. 2022. Tight Fine-Grained Bounds for Direct Access on Join Queries. *CoRR* abs/2201.02401 (2022). arXiv:2201.02401 https://arxiv.org/abs/2201.02401

[11] Karl Bringmann, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. 2020. Tree Edit Distance Cannot be Computed in Strongly Subcubic Time (Unless APSP Can). *ACM Trans. Algorithms* 16, 4 (2020), 48:1–48:22. https://doi.org/10.1145/3381878

[12] Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. 2021. Tractable Orders for Direct Access to Ranked Answers of Conjunctive Queries. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 325–341.

[13] 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 *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 393–409.

[14] Martin Grohe and Dániel Marx. 2014. Constraint Solving via Fractional Edge Covers. *ACM Trans. Algorithms* 11, 1 (2014), 4:1–4:20. https://doi.org/10.1145/2636918

[15] Jens Keppeler. 2020. *Answering Conjunctive Queries and FO+MOD Queries under Updates.* Ph. D. Dissertation. Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät. https://doi.org/10.18452/21483

[16] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. 2016. Higher Lower Bounds from the 3SUM Conjecture. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, Robert Krauthgamer (Ed.). SIAM, 1272–1287. https://doi.org/10.1137/1.9781611974331.ch89

[17] Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. 2018. Tight Hardness for Shortest Cycles and Paths in Sparse Graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, Artur Czumaj (Ed.). SIAM, 1236–1252. https://doi.org/10.1137/1.9781611975031.80

[18] Michael Mitzenmacher and Eli Upfal. 2017. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis.* Cambridge university press.

[19] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2012. Worst-case optimal join algorithms: [extended abstract]. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini (Eds.). ACM, 37–48. https://doi.org/10.1145/2213556.2213565

[20] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2018. Worst-case Optimal Join Algorithms. *J. ACM* 65, 3 (2018), 16:1–16:40. https://doi.org/10.1145/3180143

[21] Todd L. Veldhuizen. 2014. Triejoin: A Simple, Worst-Case Optimal Join Algorithm. In *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*, Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy (Eds.). OpenProceedings.org, 96–106. https://doi.org/10.5441/002/icdt.2014.13

[22] Virginia Vassilevska Williams. 2018. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*. World Scientific, 3447–3487.

[23] Virginia Vassilevska Williams and Ryan Williams. 2013. Finding, Minimizing, and Counting Weighted Subgraphs. *SIAM J. Comput.* 42, 3 (2013), 831–854. https://doi.org/10.1137/09076619X

[24] Virginia Vassilevska Williams and R. Ryan Williams. 2018. Subcubic Equivalences Between Path, Matrix, and Triangle Problems. *J. ACM* 65, 5 (2018), 27:1–27:38. https://doi.org/10.1145/3186893

[25] Virginia Vassilevska Williams and Yinzhan Xu. 2020. Monochromatic Triangles, Triangle Listing and APSP. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, Sandy Irani (Ed.). IEEE, 786–797. https://doi.org/10.1109/FOCS46700.2020.00078