The Complexity of Boolean Conjunctive Queries with Intersection Joins

Mahmoud Abo Khamis
RelationalAI
Berkeley, United States

Antonia Kormpa
University of Oxford
Oxford, United Kingdom

George Chichirim
University of Oxford
Oxford, United Kingdom

Dan Olteanu
University of Zurich
Zurich, Switzerland

ABSTRACT

Intersection joins over interval data are relevant in spatial and temporal data settings. A set of intervals join if their intersection is non-empty. In case of point intervals, the intersection join becomes the standard equality join.

We establish the complexity of Boolean conjunctive queries with intersection joins by a many-one equivalence to disjunctions of Boolean conjunctive queries with equality joins. The complexity of any query with intersection joins is that of the hardest query with equality joins in the disjunction exhibited by our equivalence. This is captured by a new width measure called the ij-width.

We also introduce a new syntactic notion of acyclicity called iota-acyclicity to characterise the class of Boolean queries with intersection joins that admit time computation modulo a poly-logarithmic factor in the data size. Iota-acyclicity is for intersection joins what alpha-acyclicity is for equality joins. It strictly sits between gamma-acyclicity and Berge-acyclicity. The intersection join queries that are not iota-acyclic are at least as hard as the Boolean triangle query with equality joins, which is widely considered not computable in linear time.

CCS CONCEPTS

• Theory of computation → Database query processing and optimization (theory).

KEYWORDS

interval joins; spatiotemporal data; segment tree; iota acyclicity

1 INTRODUCTION

Interval data is common in spatial and temporal databases. One important type of joins on intervals is the intersection join: A set of intervals join if their intersection is non-empty. In the case of point intervals, the intersection join becomes the classical equality join.

This paper establishes the complexity of Boolean conjunctive queries with Intersection Joins (denoted by IJ). Whereas the complexity of Boolean conjunctive queries with Equality Joins (denoted by EJ) has been extensively investigated in the literature by, e.g., Marx [21] and Abo Khamis et al. [4], the complexity of the more general IJ queries remained open for decades.

The key tool aiding our investigation is a many-one equivalence of any IJ query to a disjunction of EJ queries. It uses a forward reduction (IJ-to-EJ) and a backward reduction (EJ-to-IJ), cf. Figure 1.

The forward reduction takes an IJ query Q and a database D of intervals. It reduces Q to a disjunction of EJ queries, all over the same database D of numbers represented as bitstrings. The number and size of the EJ queries only depend on the structure of Q, and the size of D is within a poly-logarithmic factor from the size of D. This forward reduction enables us to use any algorithms and associated runtime upper bounds for the EJ queries as upper bounds for the IJ query Q as well. Specifically, Q’s runtime is upper bounded by the maximum runtime upper bound among the generated EJ queries.

The backward reduction takes an EJ query Q, whose structure matches that of one of the queries obtained by the forward reduction of an IJ query Q without self-joins, and an arbitrary database D2.

Figure 1: Forward and backward reductions from Sections 4 and 5 respectively. Q is a query with intersection joins, $Q_1, \ldots, Q_{\ell}$ are queries with equality joins, D and D2 are databases of intervals, D and D2 are databases of numbers.
of numbers chosen independently from \(\tilde{D}\) and \(D\) in the forward reduction. It then reduces \(Q_i\) to an \(I\) query \(Q\) whose structure matches that of the original \(I\) query \(Q_i\), and reduces \(D_2\) to some database \(D_2\) of intervals and size \(O(|D_2|)\). The backward reduction shows that we can use any lower bounds (i.e., hardness results) on any one of the \(EJ\) queries \(\tilde{Q}_i\) constructed by the forward reduction as lower bounds on the \(IJ\) query \(Q\). Together with the upper bounds, this implies that the \(IJ\) query \(Q\) is precisely the same hardness level as the hardest \(EJ\) query \(\tilde{Q}_i\). Our forward reduction thus produces an optimal solution to \(Q\) given optimal solutions to the queries \(\tilde{Q}_i\).

The quest for the optimality of \(EJ\) computation has a long history. The submodular width has been recently established as an optimal yardstick \([4, 21]\). Let \(\Phi\) be the decision problem: Given an \(EJ\) query \(\tilde{Q} \in C\) and a database \(D\), check whether \(\tilde{Q}(D)\) is true.

A natural question is then what would be an optimality yardstick for \(IJ\) computation. We settle this question with a new width notion called the \(ij\)-width. This is the maximum submodular width \([21]\) of the \(ij\)-width. This problem is fixed-parameter tractable (FPT; with parameter the query size \(|Q|\)) if there is an algorithm solving every \(C\)-instance in time \(f(|Q|) \cdot |D|^d\) for some fixed constant \(d\) and any computable function \(f\). The problem \(BCQ(C)\) is FPT if and only if every \(Q \in C\) has bounded submodular width \([21]\). The \(ij\)-width is on par with that of cyclic \(\gamma\)-acyclicity and is implied by Berge-acyclicity. The \(ij\)-width is thus a useful yardstick \([4, 21]\). Let \(\Phi\) be the decision problem: Given an \(EJ\) query \(\tilde{Q} \in C\) and a database \(D\), check whether \(\tilde{Q}(D)\) is true.

We introduce our approach using the Boolean triangle query, where each join is an intersection join:

\[ Q_\Delta = R([A], [B]) \land S([B], [C]) \land T([A], [C]) \]

Brackets denote interval variables ranging over intervals with real-valued endpoints. The two occurrences of \([A]\) in the query denote an intersection join on \(A\). An equality join on \(A\) is expressed using the variable \(A\) without brackets.

A common approach first computes the join of two of the three relations and then joins with the third relation. The first join can take \(O(N^2)\), where \(N\) is the size of the relations. An equivalent encoding of \(Q_\Delta\) using inequality joins can be computed in time \(O(N^2 \log^2 N)\) using FAQ-AI \([2]\) (Appendix E). Our approach takes time \(O(N^{3/2} \log^3 N)\), which matches the complexity of the \(EJ\) triangle query (modulo polylog factor).

Our approach is based on a decomposition of the tensor representing an intersection join. In our example, we decompose the three joins as follows. We construct three segment trees: one for the intervals from \(R\) and \(T\) for the interval variable \([A]\), another for the intervals from \(R\) and \(S\) for the interval variable \([B]\), and the third one for the intervals from \(S\) and \(T\) for the interval variable \([C]\). The segment tree for \(O(N)\) intervals can be constructed in \(O(N \log N)\) time and has depth \(O(\log N)\). Its nodes represent intervals called segments. A segment includes its descendant segments and is partitioned by its child segments. A property of a segment tree is that each input interval can be expressed as the disjoint union of at most \(O(\log N)\) segments. The problem of checking whether two input intervals intersect now becomes the problem of finding two segments, one per interval, that lie along the same root-to-leaf path in the segment tree. There are three ways this can happen: The two segments are the same, one segment is an ancestor of the other, or the other way around; this corresponds to the possible permutations of the two segments along a path, with one permutation also including the case where the segments are the same. A further property is that each permutation can be expressed using equality joins, as explained next. We encode each node in the segment tree as a bitstring: the empty string represents the root, the strings “0” and “1” represent the left and right child respectively, the strings “00” and “01” represent the left and right child respectively, the strings “10” and “11” represent the left and right child respectively, and so on. Given two nodes \(n_1\) and \(n_2\), where \(n_1\) is an ancestor of \(n_2\), the bitstring for \(n_1\) is then a prefix of that for \(n_2\). We can capture this relationship in a query with equality joins: We use one variable \(A_1\) to stand for the bitstring for \(n_1\), which is also a prefix of the bitstring for \(n_2\), and \(A_2\) to stand for the remaining bitstring for \(n_2\).

There are eight possible configurations for the ancestor-descendant relationship between the segment tree nodes for the two \(A\)-intervals in \(R\) and \(S\) and similarly for \(B\) and \(C\). Each such case can be expressed using an \(EJ\) query. Our query \(Q_\Delta\) is then equivalent to the disjunction \(\tilde{Q}_\Delta = \bigvee_{i \in [8]} \tilde{Q}_i\) of the following \(EJ\) queries:

\[
\begin{align*}
\tilde{Q}_1 &= R_{\exists2}(A_1, A_2, B_1, B_2) \land S_{\exists2}(B_1, C_1, C_2) \land T_{\exists2}(A_1, C_1) \\
\tilde{Q}_2 &= R_{\exists2}(A_1, A_2, B_1, B_2) \land S_{\exists2}(B_1, C_1) \land T_{\exists2}(A_1, C_1, C_2) \\
\tilde{Q}_3 &= R_{\exists2}(A_1, A_2, B_1) \land S_{\exists2}(B_1, C_1, C_2) \land T_{\exists2}(A_1, C_1) \\
\end{align*}
\]
The query is defined over new relations $R_{A,B}$, $S_{B,C}$, and $T_{B,C}$ that are transformations of the original relations $R$, $S$, and $T$ to hold the bitstrings in place of the original intervals. The subscript $A = i$ stands for the indices $1, \ldots, i$ of the variables $A_1, \ldots, A_i$ ranging over bitstrings; similarly for $B$ and $C$. To avoid clutter, in the remainder of this paper we will denote all such constructed new relations $R_{A,B}$ by $\tilde{R}$ and use their schema to identify them uniquely. For input relations of size $N$, the new relations have size $O(N \log^2 N)$, with one logarithmic factor per join interval variable. We next explain the purpose of these bitstring relations. A tuple $(a_1, a_2, b_1, b_2)$ is in $R_{1,2}$ if and only if there is a pair of intervals $(a_1, a_2)$ in $R$ and the concatenations $a_1 \circ a_2$ and $b_1 \circ b_2$ reconstruct bitstrings of nodes covered by intervals $a_1$ and respectively $b_1$. Similarly, a tuple $(b_1, c_1, c_2)$ in $S_{1,2}$ reconstructs the bitstrings $b_1$ and $c_1 \circ c_2$ of nodes covered by intervals $b_1$ and $c_1$. A tuple $(a_1, c_1)$ in $T_{1,2}$ specifies the bitstrings $a_1$ and $c_1$ of nodes covered by $a_1$ and $c_1$. Therefore, $\tilde{Q}_1$ holds in case: (1) there exists a node covered by an interval $a_1$ in $R$ and a node covered by an interval $a_1$ in $T$ such that the bitstring of the latter is a prefix of the former; (2) there exists a node covered by an interval $b_1$ in $R$ and a node covered by an interval $b_2$ in $T$ such that the bitstring of the latter is a prefix of the former; and (3) there exists a node covered by an interval $c_1$ in $S$ and a node covered by an interval $c_1$ in $T$ such that the bitstring of the latter is a prefix of the former. Equivalently, $a_1$ intersects with $a_1$, $b_1$, $b_1$, and $c_1$ intersects with $c_1$, i.e., $R(a_1, a_2) \land S(b_1, b_2) \land T(c_1, c_2)$ holds.

The hypergraph of each of the eight $EJ$ queries admits a hypertree decomposition in the form of a star with the central bag $\{A_1, B_1, C_1\}$, cf. Figure 2. In each of these decompositions, the materialisation of this bag requires solving the triangle join $R'(A_1, B_1) \land S'(B_1, C_1) \land T'(A_1, C_1)$, where $R'$ is a projection of $R_{A,B}$ to $A_1, B_1$ and similarly for $S'$ and $T'$. The new relations and their projections have size $O(N \log^2 N)$. The materialisation of the join takes time $O((N \log^2 N)^{3/2}) = O(N^{3/2} \log^2 N)$ using existing worst-case optimal join algorithms [22]. Checking whether any of the eight $EJ$ queries is true takes time linear in the maximum size of the bags of its decomposition. This gives an overall computation time $O(N^{3/2} \log^2 N)$ for $\tilde{Q}_A$ and also for $\tilde{Q}_B$.

We close the example with a discussion on an alternative encoding: Instead of $\tilde{R}(A_1, A_2, B_1, B_2)$, we can use its lossless decomposition into $\tilde{R}_A(Id, A_1, A_2)$ and $\tilde{R}_B(Id, B_1, B_2)$. Here, a tuple $(i, a_1, a_2) \in \tilde{R}_A$ encodes that the $[A]$-interval in the tuple of $R$ with identifier $i$ is mapped to the bitstring $a_1 \circ a_2$ in the segment tree for $[A]$. We thus avoid the explicit materialisation of all combinations of encodings for $[A]$ and $[B]$ (and the same for the other two pairs of variables). This decomposition applies systematically to all constructed relations. Each of these new relations has size $O(N \log N)$, which is less than $O(N \log^2 N)$ in our default encoding. In general, for each $m$-way join interval variable, this encoding creates $m$ new relations regardless of whether such variables occur in the same relational atom in the query. In contrast, our default encoding creates $m^k$ new relations for each atom that contains $k$ such $m$-way join variables. Although more space efficient, this encoding comes with the same data complexity (modulo log factors) as the one used in the paper.

\section{Related Work}

Algorithms for intersection joins have been developed in the context of temporal [11] and spatial databases [13, 18]. In temporal databases, tuples can be associated with intervals that represent the valid time periods. Temporal or interval joins are used to match tuples that are valid at the same time. In spatial databases, tuples can be associated with 2D objects that are approximated by two intervals defining minimum bounding rectangles [19]. Spatial joins are used to find tuples with overlapping bounding rectangles. Temporal and spatial joins are thus intersection joins. Similarity joins under different distance metrics can be reduced to geometric containment, which is expressible using intersection joins [12].

Intersection joins. There is a wealth of work on algorithms for intersection joins, mostly binary joins computed one at a time and over relations with 1D or 2D intervals [18]. These algorithms use indices or partitioning and are typically disk-based with the objective of minimising I/O accesses. Examples of index-based algorithms include: the slot index spatial join [20], the seeded tree join [16], the R-tree join [8], and relational interval tree join [9]. Extensions of binary joins [8] to multi-way joins have also been considered [19]. Partition-based algorithms include: the partition based spatial-merge join [24], the spatial hash join [17], the size separation spatial join [15], the sweeping-based spatial join [5], and the plane-sweep method [27]. The partition-based algorithms can be naturally parallelised and distributed [6, 26, 28]. These algorithms...
can compute two-way intersection joins in $O(N \log N + OUT)$, where OUT is the output size and $N$ is the input size. To sum up, there is no development on optimal algorithms for queries with intersection joins. Furthermore, most existing approaches focus on one join at a time, which can be suboptimal since they can produce intermediate results asymptotically larger than the final result, as in the case of equality joins [23]. Our approach escapes the limitation of existing intersection join algorithms and benefits from worst-case optimal algorithms for equality joins.

**Inequality joins.** An intersection join can be expressed as a disjunction of inequality joins: Given two intervals $[l_1, r_1]$ and $[l_2, r_2]$ defined by their starting and ending points, checking whether they intersect can be expressed as $(l_1 \leq l_2 \leq r_1) \lor (l_2 < l_1 \leq r_2)$. If queries can thus be reformulated in the framework of Functional Aggregate Queries with Additive Inequalities (FAQ-AI) [2]. The hypergraph of an FAQ-AI has two types of hyperedges: normal hyperedges, one per relation in the query and that covers the nodes representing the variables of that relation, and relaxed hyperedges, one per inequality join and that covers the variables in the inequality. This hypergraph is subject to relaxed hypertree decompositions, which are fractional hypertree decompositions [21] where each normal hyperedge is covered by one bag and each relaxed hyperedge is covered by two adjacent bags. For a database of size $N$, an FAQ-AI can be solved in time $O(N^{\text{subw}} \cdot \text{polylog } N)$, where subw is the relaxed submodular width of the FAQ-AI and corresponds to the submodular width of the FAQ-AI hypergraph computed over its possible relaxed hypertree decompositions [2]. For the triangle $\langle I \rangle$ query $Q_3$ in Section 1.1, the ij-width is $ijw(Q_3) = 3/2$ yet subw$_w(Q_3) = 2$ (Appendix E). Furthermore, it can be shown that $ijw$ is lower than subw$_w$ for the Loomis-Whitney 4, and the 4-clique $\langle I \rangle$ queries (see Table 1 for a summary and [14] for a full analysis).

3 PRELIMINARIES

This section introduces notation used in the main body of the paper. For lack of space, further preliminaries and proofs are deferred to Appendix A and the full version of the paper [14].

**Segment Tree.** Let $I$ be a set of $n$ intervals. Let $p_1, \ldots, p_m$ be the sequence of the distinct endpoints of the intervals in ascending order (so, $m \leq 2n$). Consider the following disjoint intervals called elementary segments that form a partition of the real line: $(-\infty, p_1)$, $(p_1, p_1)$, $(p_1, p_2)$, $(p_2, p_2)$, $\ldots$, $(p_{m-1}, p_m)$, $(p_m, p_m)$, $(p_m, +\infty)$. The segment tree $\mathcal{I}_I$ for $I$ is a complete binary tree $^1$, where:

- The leaves of $\mathcal{I}_I$ correspond to the elementary segments induced by an order of the endpoints of the intervals in $I$: the leftmost leaf corresponds to the leftmost elementary segment, and so on. The elementary segment corresponding to a leaf $\nu$ is denoted by $\nu(v)$.
- The internal nodes of $\mathcal{I}_I$ correspond to segments that are the union of elementary segments at the leaves of their subtrees: the segment $\nu(v)$ corresponding to an internal node $v$ is the union of the elementary segments $\nu(u)$ at the leaves $u$ in the subtree rooted at $v$; $\nu(u)$ is thus the union of the segments at its two children.

---

$^1$In a complete binary tree, every level, except possibly the last, is completely filled and the nodes in the last level are positioned as far left as possible. Every node of the segment tree is thus either a leaf or an internal node with exactly two children.

---

![Figure 3: Segment tree on the set of intervals $I = \{\square = [1, 4], \bullet = [3, 4]\}$. The interval $[1, 4]$ is contained in the canonical subsets of the nodes $001$, $01$, and $10$. The interval $[3, 4]$ is contained in the canonical subsets of the nodes $011$ and $10$.](image)
whose values are real numbers from a finite domain \( \text{Dom}(X) \). For an interval \( x \), we use \( x.l \) and \( x.r \) to denote its left and respectively right endpoints. Given a set \( e \) of variables, \( R_e \subseteq \prod_{X \in e} \text{Dom}(X) \) is a relation consisting of tuples of \( |e| \) real values; relations over intervals are defined similarly by replacing \( X \) with \{ \( X \) \}. A tuple \( t \) with schema \( e \) is a mapping of the variables in \( e \) to values in their domains. We denote by \( t(X) \) the value for variable \( X \) (or \{ \( X \) \}) in \( t \) and by \( t(e') \) the set mapping variables in \( e' \subseteq e \) to their values in \( t \).

A \((multi-)\)hypergraph \( \mathcal{H} = (V, E) \) has a set \( V \) of vertices and a multiset \( E \subseteq 2^{\mathcal{V}} \) of hyperedges. We label the hyperedges to distinguish between those representing the same set of vertices. For a vertex \( X \in V, E_X \) denotes the subset of \( E \) that contains \( X \).

**Definition 3.3 (Queries).** Given a hypergraph \( \mathcal{H} = (V, E) \), where \( V \) is a set of variables, a query over \( \mathcal{H} \) has the form \( Q = \bigwedge_{e \in E} R_e(e) \). If the vertices in \( \mathcal{H} \) are interval variables, then \( Q \) is a Boolean conjunctive query with intersection joins, or \( \mathcal{IJ} \) for short. If the vertices in \( \mathcal{H} \) are point variables, then \( Q \) is a Boolean conjunctive query with intersection and equality joins, or \( \mathcal{EI} \) for short.

An \( \mathcal{EI} \) query \( Q = \prod_{e \in E} R_e(e) \) evaluates to true if and only if there exist tuples \( (t_e)_{e \in E} \in \prod_{e \in E} R_e \) that satisfy the following:

- \( \forall X \in V, t_e(X) \neq \emptyset \).
- \( \forall X \in V, t_e(X) = \{ X \} \) for all \( e \in E_X \).

For the evaluation of an \( \mathcal{IJ} \) query with hypergraph \( \mathcal{H} = (V, E) \) over a database \( D \), we assume without loss of generality that the schema of \( D \) is given by \( \mathcal{H} \); any database can be brought into this form by appropriately ensuring a bijection between the vertices in \( V \) and attributes in \( D \) and a bijection between the hyperedges \( e \in E \) and the relations \( R_e \) over schema \( e \) in \( D \).

Given a set \( S \), a permutation of \( S \) is an ordered sequence of the elements in \( S \). We denote by \( \pi(S) \) the set of all permutations of the elements in \( S \). For a sequence \( s \), \( s_j \) denotes its \( j \)-th element. The concatenation of sequences \( s_1, \ldots, s_k \) is denoted by \( s_1 \circ \cdots \circ s_k \).

### 4 FROM INTERSECTIONS TO EQUALITIES

In this section, we show that the \( \mathcal{IJ} \) evaluation problem can be reduced to the \( \mathcal{EI} \) evaluation problem. This forward reduction is used to give an upper bound on the time complexity for the former problem using the complexity of the latter problem. Section 5 then presents a backward reduction to give a corresponding lower bound on the time complexity of the \( \mathcal{IJ} \) evaluation problem (recall Figure 1).

#### 4.1 Rewriting the Intersection Predicate

At the core of \( \mathcal{IJ} \) evaluation lies the non-emptiness check of the intersection of \( k \) intervals \( x_1, \ldots, x_k \in I \). We call this check the intersection predicate. In this section, we show how to rewrite this predicate into an equivalent form that uses the canonical partitions of the intervals in a segment tree \( \mathcal{T}_I \).

Since the elementary segments that correspond to the leaves of \( \mathcal{T}_I \) form a partition of \( \mathbb{R} \), for any point \( p \in \mathbb{R} \) there is precisely one leaf node \( \text{leaf}(p) \) such that \( p \in \text{seg} \)(\( \text{leaf}(p) \)). By Property 3.2(1), \( \text{anc}(\text{leaf}(p)) = \{ o \in V(\mathcal{T}_I) \mid p \in \text{seg}(o) \} \). That is, the nodes whose segments contain the point \( p \) are precisely the ancestors of \( \text{leaf}(p) \).

**Lemma 4.1 (Intersection Predicate Rewriting 1).** For any set of intervals \( S = \{ x_1, \ldots, x_k \} \subseteq I \), the predicate \( \left( \bigcap_{i \in [k]} x_i \right) \neq \emptyset \) is equivalent to:

\[
\bigvee_{\sigma \in \pi(S)} \bigg( \bigvee_{(u_1, \ldots, u_k) \in \text{anc}(\text{leaf}(x_i))} \bigg( \bigwedge_{j \in [k-1]} u_j \in \text{CP}_I(\sigma_j) \bigg) \bigg)
\]

Lemma 4.1 states the following. The intervals in \( S \) intersect if and only if there is an interval \( x_i \in S \) such that the canonical partitions of each other interval in \( S \) contain an ancestor of \( \text{leaf}(x_i) \).

By construction, this leaf contains the left endpoint of \( x_i \).

**Property 4.2.** Consider a set of intervals \( S = \{ x_1, \ldots, x_k \} \subseteq I \) and a segment tree \( \mathcal{T}_I \). For any \( x_i \in S \), there can be at most one tuple of nodes \( v_j \in \text{anc}(\text{leaf}(x_i)) \) for \( j \in [k], j \neq i \) that satisfies the conjunction of Lemma 4.1.

The conjunction in Lemma 4.1 can be satisfied by several \( i \)-values when there are several intervals in \( S \) that have the same left endpoint. The database can be transformed such that any two intervals from different relations have distinct left endpoints without affecting query evaluation [14]. If the intervals in \( S \) have distinct left endpoints, then the conjunction in Lemma 4.1 can be satisfied by at most one \( i \in [k] \), namely the one with the maximum left endpoint of the intervals in \( S \); this is also the left endpoint of the interval representing the intersection of all intervals in \( S \).

Consider a path from the root of the segment tree \( \mathcal{T}_I \) down to \( \text{leaf}(x_i) \) from Lemma 4.1. The nodes \( v_1, \ldots, v_k \) from Lemma 4.1 all lie on this path. These nodes satisfy \( u_j \in \text{CP}_I(x_i) \) for all \( j \in [k] - \{i\} \).

**Lemma 4.3 (Intersection Predicate Rewriting 2).** For any set of intervals \( S = \{ x_1, \ldots, x_k \} \subseteq I \), the predicate \( \left( \bigcap_{i \in [k]} x_i \right) \neq \emptyset \) is equivalent to:

\[
\bigvee_{\sigma \in \pi(S)} \bigg( \bigvee_{(u_1, \ldots, u_k) \in \text{anc}(\text{leaf}(x_i))} \bigg( \bigwedge_{j \in [k-1]} u_j \in \text{CP}_I(\sigma_j) \bigg) \bigg)
\]

By Property 4.2, for a permutation \( \sigma \in \pi(S) \), there can be at most one tuple \( (u_j)_{j \in [k-1]} \) that satisfies the conjunction. However, even if the intervals in \( S \) have distinct left endpoints, the predicate of Lemma 4.3 may be satisfied by multiple permutations. To see this, suppose that \( \sigma \) and \( (u_j)_{j \in [k-1]} \) satisfy the predicate. If there is \( j \in [k-1] \) such that \( u_j = u_{j+1} \), then the permutation \( \sigma' \), obtained
by swapping $\sigma_j$ and $\sigma_{j+1}$ in $\sigma$, together with the tuple $(u_j)_{j \in [k-1]}$, also satisfy the predicate. It is possible to further restrict the permutations such that each tuple of segment tree nodes that satisfies the conjunction corresponds to exactly one permutation [14]. The equivalence in Lemma 4.3 can be alternatively expressed using the bitstrings of the nodes in the segment tree. By Property 3.2 (1) the expression $u_j \in \text{anc}(u_{j+1})$ can be equivalently stated as $u_j$ being a prefix of $u_{j+1}$. In other words, there exists a tuple of bitstrings $(b_1, \ldots, b_k)$ such that $u_j = b_1 \cdots b_j$ for $j \in [k]$. This observation leads to the following rewrite of Lemma 4.3.

**Lemma 4.4 (Intersection Predicate Rewriting 3).** Consider a set of intervals $S = \{x_1, \ldots, x_k\} \subseteq I$. The predicate $(\bigcap_{x \in S} x) \neq \emptyset$ is true if and only if there exists a permutation $\pi(S)$ and a tuple of bitstrings $(b_1, \ldots, b_k)$ such that:

- $(b_1 \cdots b_{j-1}) \in CP_j(\sigma_j)$ for $j \in [k]$, and
- $(b_1 \cdots b_j) = \text{leaf}(\sigma_j)$ for $j = k$.

Section 4.2 lifts the rewriting in Lemma 4.4 to the level of queries.

### 4.2 One-Step Forward Reduction

For an $\mathcal{L}$ query $Q$ with hypergraph $H = (V, E)$ and a database $D$, the forward reduction proceeds iteratively on $Q$ and $D$ and resolves one join interval variable at a time. Let this variable be $[X]$. The reduction yields a new query $\hat{Q}(Q_D)$ and a new database $D_{[X]}$ such that $Q([X]) (D_{[X]})$ is true if and only if $Q(D)$ is true.

The core computation needed to evaluate $Q$ over $D$ is the intersection predicate $(\bigcap_{x \in S} x) \neq \emptyset$, where $S$ consists of one input interval per relation involved in the intersection join on $[X]$. Lemma 4.4 explains how to express this computation for any subset $S$ of an input set of intervals $I$ for $[X]$ using the segment tree $\mathcal{T}_I$ for $I$.

Let $k = |E[X]|$ be the number of hyperedges in $H$ containing $[X]$. The reduction maps $[X]$ to fresh point variables $X_1, \ldots, X_k$ that range over all the possible bitstrings of the segment tree nodes from the canonical partitions of the intervals of $[X]$.

Given a permutation $\sigma = (\sigma_1, \ldots, \sigma_k) \in \pi(E_{[X]}))$ of the hyperedges $E_{[X]}$ containing $[X]$, each hyperedge $\sigma_i$ induces a fresh hyperedge $\hat{\sigma}_i$ that has the fresh point variables $X_{\sigma_1}, \ldots, X_{\sigma_k}$ in place of the original interval variable $[X]$; $\hat{\sigma}_i \equiv \sigma_i \setminus \{[X]\} \cup \{X_{\sigma_1}, \ldots, X_{\sigma_k}\}$.

**Definition 4.5 (One-Step Hypergraph Transformation).** Given a hypergraph $H = (V, E)$, an interval variable $[X]$, and any permutation $\sigma \in \pi(E_{[X]}))$, the hypergraph $H_{[X]}(\sigma)$ has the set $V_{[X]}$ of vertices and the set $E_{[X]}(\sigma)$ of hyperedges, where $V_{[X]} = \{X\} \cup \{X_{\sigma_1}, \ldots, X_{\sigma_k}\}$ and $E_{[X]}(\sigma) = E \setminus \{\sigma_i \mid i \in [k]\} \cup \{\hat{\sigma}_i \mid i \in [k]\}$. The set $H_{[X]} = \{H_{[X]}(\sigma) \mid \sigma \in \pi(E_{[X]}))\}$ consists of all hypergraphs created from $H$ by resolving the interval variable $[X]$.

For a given permutation $\sigma$, there is a one-to-one correspondence between the hyperedges in $H$ and those in $H_{[X]}(\sigma)$. We obtain as many new hypergraphs as permutations of $E_{[X]}$.

**Example 4.6.** Let $H$ be a hypergraph with vertices $\{[A], [B], [C]\}$ and edges $e_1 = e_2 = \{[A], [B], [C]\}$ and $e_3 = \{[A]\}$.

We reduce $H$ by resolving the interval variable $[A]$. Since $[A]$ occurs in three edges, we create three point variables $A_1, A_2, A_3$ and consider six permutations. The new edges created for the permutation $\sigma = (e_1, e_2, e_3)$ are: $\hat{\sigma}_1 = \{A_1, [B], [C]\}$, $\hat{\sigma}_2 = \{A_1, A_2, [B], [C]\}$, and $\hat{\sigma}_3 = \{A_1, A_2, A_3\}$. For the permutation $(e_3, e_2, e_1)$, the new edges are: $\{A_1\}$, $\{A_1, A_2, [B], [C]\}$, and $\{A_1, A_2, A_3, [B], [C]\}$.

Each hypergraph $H_{[X]}(\sigma)$ defines a new query $Q_{[X]},(\sigma)$ and the corresponding database $D_{[X]}(\sigma)$. We next explain how to construct the new query and the new database.

The query $Q$ is rewritten according to the new hypergraphs: For each permutation $\sigma$, we create an EL query $\hat{Q}_{[X]}(\sigma)$, which has equality joins and possibly remaining intersection joins, whose hypergraph is $H_{[X]}(\sigma)$. By taking all permutations, we thus create a query $\hat{Q}_{[X]}$ that is a disjunction of EL queries such that each such query has one join interval variable less, namely $[X]$. Note that $Q$ need not be an EL query: it may have both intersection and equality joins, for instance if it is the result of a previous rewriting step.

**Definition 4.7 (One-Step Query Rewriting).** For any EL query $Q$ with hypergraph $H = (V, E)$, the EL query $\hat{Q}_{[X]}(\sigma)$ with hypergraph $H_{[X]}(\sigma) = (V_{[X]}, E_{[X]}(\sigma))$ is defined by:

$$\hat{Q}_{[X]}(\sigma) = \bigvee_{\sigma \in \pi(E_{[X]}))} \hat{R}(\sigma)$$

The query $\hat{Q}_{[X]}$ is the disjunction of the EL queries $\hat{Q}_{[X]}(\sigma)$ over all possible permutations $\sigma \in \pi(E_{[X]}))$.

**Example 4.8.** The EL query $Q = R([A], [B], [C]) \land S([A], [B], [C]) \land \tau([A])$ has the hypergraph $H$ in Example 4.6. The permutation $(e_1, e_2, e_3)$ yields $\hat{Q} = R(A_1, [B], [C]), \hat{S}(A_1, A_2, [B], [C]), \hat{\tau}(A_1, A_2, A_3)$. The permutation $(e_3, e_2, e_1)$ yields $\hat{Q} = R(A_1, A_2, A_3, [B], [C]), \hat{S}(A_1, A_2, [B], [C]), \hat{\tau}(A_1)$. The final query $\hat{Q}_{[A]}$ is a disjunction of six EL queries, including $\hat{Q}_1$ and $\hat{Q}_2$.

For each new hyperedge $\hat{\sigma}_i$ in $H_{[X]}(\sigma)$, there is a new relation $\hat{R}_{\sigma}([\hat{\sigma}_i])$. To avoid clutter, we denote it by $\hat{R}(\hat{\sigma}_i)$; its schema $\hat{\sigma}_i$ uniquely identifies the transformation of the original relation $R$.

**Definition 4.9 (One-Step Database Transformation).** The database $D_{[X]}(\sigma)$ is constructed from the database $D$ as follows. For each tuple $t \in R(\sigma_i)$, we construct tuples $t \in \hat{R}(\hat{\sigma}_i)$ such that:

- $(t(\sigma_i \setminus \{i\})) = (t(\hat{\sigma}_i \setminus \{X_{\sigma_1}, \ldots, X_{\sigma_k}\}))$
- $(i(X_1) \circ \cdots \circ i(X_k)) = CP_j \tau(t([X]))$ if $i \in [k-1]$
- $(i(X_1) \circ \cdots \circ i(X_k)) = \text{leaf}(t([X]))$ if $i = k$

The relations whose schemas do not contain $[X]$ are copied from $D$ to $D_{[X]}(\sigma)$. The new database $D_{[X]}(\sigma)$ is the set of all relations in the databases $D_{[X]}(\sigma)$.

The number of tuples $t$ constructed for a tuple $t$ in Definition 4.9 depends on the size of the canonical partition $CP_j \tau(t([X]))$ of $t([X])$ and on the number of ways we can partition the bitstring of a node in the canonical partition into $i$ substrings. Overall, this number is poly-logarithmic in the number of input intervals $I$. This is made more precise in the next lemma.

**Lemma 4.10.** Each new relation $\hat{R}(\hat{\sigma}_i)$ in database $D_{[X]}(\sigma)$ constructed from the database $D$ following Definition 4.9 has the size $O(|\{R(\sigma_i)\}| \cdot \log^i |I|)$ if $i \in [k-1]$ and $O(|R(\sigma_i)| \cdot \log^{k-1} |I|)$ if $i = k$. It can be constructed in time proportional to its size.
The transformations in Definitions 4.9 and 4.7 preserve the equivalence to the original evaluation problem: The result of $Q$ over $D$ is the same as the result of $\hat{Q}(X)$ over $\hat{D}(X)$.

**Lemma 4.11.** Given any $E|J$ query $Q$, interval variable $[X]$ in $Q$, and any database $D$, let the $E|J$ query $\hat{Q}(X)$ and database $\hat{D}(X)$ be constructed as per Definitions 4.9 and 4.7. Then, $Q(D)$ is true if and only if $\hat{Q}(X)\hat{D}(X)$ is true.

**Example 4.12.** We demonstrate the database transformation for the query $Q_3$ in Section 1.1. Consider the interval variable $[A]$. We convert $Q_3$ into the disjunction of two queries with two new point variables $A_1$ and $A_2$ in lieu of the interval variable $[A]:$

\[
\hat{Q}' = \tilde{R}_1(A_1, [B]) \land S([B], [C]) \land \tilde{T}_2(A_1, A_2, [C]),
\]

\[
\hat{Q}'' = \tilde{R}_2(A_1, A_2, [B]) \land S([B], [C]) \land \tilde{T}_2(A_1, [C]).
\]

Let $D$ be the input database with relations $R, S$ and $T$ and let $N$ be the size of $D$. By Definition 4.9, we construct a new database instance $\hat{D}[A]$ that consists of $S$ and four new relations $\tilde{R}_1, \tilde{T}_1, \tilde{T}_2$ and $\tilde{T}_3$. By Lemma 4.11, $Q_3(D)$ is true if and only if $\hat{Q}'(\hat{D}(A))$ or $\hat{Q}''(\hat{D}(A))$ are true.

Let $I$ be the set of all $[A]$-intervals in $R$ and $T$. We have $|I| \leq 2N$. Let $E$ be the set of endpoints of these intervals. We have $|E| \leq 2|I|$. Assume that $E = \{0, 1, 2, \ldots, k\}$ for some integer $k \leq 2N$. This is without loss of generality since the intersection problem does not depend on the absolute values of the end points but only on their relative positioning. Assume also that $k$ is a power of 2; otherwise, replace $k$ with the smallest power of 2 that is $\geq k$. Let $I_E$ be the segment tree whose root corresponds to the interval $[0, k]$, and the left and right children of each node correspond to the left and right halves of the corresponding interval respectively. Assume without loss of generality that each interval $i \in I$ corresponds to a node in $I_E$; otherwise, $i$ can be broken down into $O(\log_2(k)) = O(\log_2 N)$ intervals that correspond to nodes in $I_E$ (Property 3.2 (3)). Each node $n$ in $I_E$ can be encoded as a binary string, cf. Section 3:

If $n$ corresponds to the string $b$, then its left and right children correspond to the strings $b0$ and $b1$ respectively. For an interval $i$ that corresponds to $n$ in $I_E$, let $\text{bin}(i)$ denote the binary string that encodes node $n$. Let $\tilde{t}_1$ and $\tilde{t}_2$ be two intervals that correspond to the nodes $n_1$ and $n_2$. They intersect if and only if one of the two nodes is an ancestor of the other (or the nodes are the same). That is, one of them contains the other (Property 3.3, part (1)). Equivalently, they intersect if and only if one of the two binary strings $\text{bin}(i_1)$ and $\text{bin}(i_2)$ is a prefix of the other (Property 3.2 (1)).

Define the new relations as follows (Definition 4.9):

\[
\tilde{R}_1 = \{(\text{bin}([a]), [b]) \mid ([a], [b]) \in R\},
\]

\[
\tilde{T}_1 = \{(\text{bin}([a], c)) \mid ([a], [c]) \in T\},
\]

\[
\tilde{T}_2 = \{(a_1, a_2, [b]) \mid ([a], [b]) \in R \land a_1 \circ a_2 = \text{bin}([a])\},
\]

\[
\tilde{T}_3 = \{(a_1, a_2, [c]) \mid ([a], [c]) \in T \land a_1 \circ a_2 = \text{bin}([a])\}.
\]

By Lemma 4.10 and the assumption that each $[A]$-interval corresponds to exactly one node of $I_E$, we have $|\tilde{R}_1| = |\tilde{R}_2| = |\tilde{T}_1|$, and both $\tilde{T}_1$ and $\tilde{T}_2$ can be constructed in linear time. Furthermore, $|\tilde{T}_2| = O(|T| \cdot \log N)$ because there are $O(\log N)$ ways to break a binary string $\text{bin}(a)$ of length $O(\log N)$ in two. $\tilde{T}_1$ can be constructed in time $O(|T| \cdot \log N)$. Similarly for $\tilde{R}_2$ (Lemma 4.10).

Then, $\hat{Q}'(\hat{D}(A))$ holds if and only if $Q(D)$ has a satisfying assignment where the $[A]$-interval from $R$ contains the $[A]$-interval from $T$. Also, $\hat{Q}''(\hat{D}(A))$ holds if and only if $\hat{Q}(D)$ has a satisfying assignment where the $[A]$-interval from $R$ is contained in the $[A]$-interval from $T$. Consequently $Q(D)$ holds if and only if $\hat{Q}'(\hat{D}(A))$ or $\hat{Q}''(\hat{D}(A))$ holds (Lemma 4.11).

**4.3 Full Forward Reduction**

In this section, we show how to completely reduce (1) any $IJ$ query to a disjunction of $E|J$ queries and (2) any database to a database with bitstrings in place of intervals for join interval variables. In particular, the full reduction is obtained by iteratively applying the reduction step from Section 4.2 for each interval variable to the result of the previous reduction step or to the input query and database in case of the first reduction step.

**Algorithm 1** $IJ$ to $E|J$ Reduction

**Input:** $I$ query $Q$ with hypergraph $\mathcal{H}$, database $D$

```
1: procedure REDUCE($H = \{\mathcal{H}\}, Q = \{Q\}, D = D$)
2:   for each interval join variable $[X]$ in $Q$ do
3:       $H_0 := H, Q := \emptyset$
4:       for each $E \in H_0$ do
5:           create $\mathcal{H}(X)$ from $\mathcal{H}$ following Definition 4.5
6:           $H := H \cup \mathcal{H}(X)$
7:       $Q_0 := Q, \tilde{Q} := \emptyset$
8:       for each $Q \in Q_0$ do
9:           for each $\sigma \in \pi(E[X])$ do
10:              create $\tilde{Q}(X), Y$ from $Q$ following Definition 4.7
11:                 $\tilde{Q} := \tilde{Q} \cup \tilde{Q}(X, \sigma))$
12:                 $\tilde{D}[X] := \tilde{D}[X]$
13:                 $\tilde{D} := \tilde{D}[X]$
14:   return $(H, \tilde{Q})$
```

Algorithm 1 details the reduction. The result is a triple consisting of: the set $H$ of hypergraphs constructed by iteratively resolving the join interval variables in the input $IJ$ query $Q$; the set $Q$ of $E|J$ queries, with one such query per hypergraph in $H$; and the database $D$. The final query is the disjunction of the $E|J$ queries in $Q$.

We further define the transformation function $\tau$ that takes any hypergraph $\mathcal{H}$ to the set of hypergraphs $\mathcal{H}: \tau(\mathcal{H}) = H$, where $(H, Q, D) = \text{REDUCE}(\mathcal{H}, Q, D)$. This is used in the following sections to define the complexity of $IJ$ queries.

The next theorem states that our reduction is correct.

**Theorem 4.13 (Correctness).** For any $IJ$ query $Q$ with hypergraph $\mathcal{H}$ and any database $D$, it holds that $Q(D)$ is true if and only if $\tilde{Q}(\hat{D})$ is true, where $(H, Q, D) = \text{REDUCE}(\mathcal{H}, Q, D)$.

**4.4 Complexity of $IJ$ Queries**

We give the data complexity of $IJ$ queries using the reduction to $E|J$ queries from Section 4.3. We next define a new width measure for $IJ$ queries called the $ij$-width using the submodular width of the $E|J$ queries obtained in the full reduction (Definition A.9).
Session 2: Query Evaluation

| IJ Query | FAQ-AI [2] | Our approach |
|----------|----------|--------------|
| Triangle | $O(N^2 \log^3 N)$ | $O(N^3/2 \log^3 N)$ |
| Loomis-Whitney-4 | $O(N^2 \log^3 N)$, for $k \geq 9$ | $O(N^{5/3} \log^3 N)$ |
| 4-clique | $O(N^3 \log^3 N)$, for $k \geq 5$ | $O(N^2 \log^3 N)$ |

Table 1: Our approach versus FAQ-AI for three IJ queries.

**Definition 4.14 (ij-width).** For any hypergraph $H$, the ij-width of $H$ is defined as follows:

$$ijw(H) = \max_{H \in r(H)} \text{subw}(H)$$

The complexity of a given IJ query is that of the most expensive EJ query constructed by the full reduction. This justifies taking the maximum in the definition of the ij-width. The optimality yardstick for the evaluation of EJ queries is given by the submodular width [21] (see also discussion in Section 1), which justifies the use of this width measure in the definition of the ij-width.

Let $\mathcal{V}^H \subseteq \mathcal{V}$ be the set of join interval variables in the query and $\mathcal{E}^H = |E^H|$ be the number of hyperedges containing the interval variable $X$. Our full reduction constructs up to $|\mathcal{V}^H| \cdot |\mathcal{E}^H|$ new relations and up to $|\mathcal{V}^H| \cdot |\mathcal{E}^H|!$ EJ queries. By Lemma 4.10, each new relation has size $O(N \log N)$, where $N$ is the size of the input relations. The number of constructed EJ queries only depends on the structure of $Q$.

**Theorem 4.15.** Given any IJ query $Q$ with hypergraph $H$ and database $D$, $Q(D)$ can be computed in time $O(|D|^{ijw(H)} \cdot \text{polylog}|D|)$. Table 1 gives the time complexities for FAQ-AI [2] and our approach for the cyclic IJ queries: triangle, Loomis Whitney with 4 variables, and 4-clique. Appendix E details the FAQ-AI evaluation of a reformulation of the triangle IJ query using inequality joins.

**5 FROM EQUALITIES TO INTERSECTIONS: REDUCTION OPTIMALITY**

In the previous section, we showed how to reduce the evaluation of an IJ query $Q$ over a database $D$ of intervals to a disjunction $\mathcal{Q}$ of EJ queries over a database $D$ of numbers, where $|D| = O(|D| \cdot \text{polylog}|D|)$. This proves that the runtime on $Q$ is upper bounded by the maximum upper bound over all queries in $\mathcal{Q}$ (within a polylog factor). In this section, we do the opposite. We show that the runtime on $Q$ is also lower bounded by the maximum lower bound over all queries in $\mathcal{Q}$. We start with an EJ query $\mathcal{Q}$, whose query structure matches that of one of the queries in $\mathcal{Q}$, and with an arbitrary database $D_2$ over the schema of $\mathcal{Q}$. The values in $D_2$ are numbers and can be chosen independently from $D$ and $D_1$ in the forward reduction. We show how to reduce $\mathcal{Q}(D_2)$ to $Q(D_1)$ when $Q$ is an IJ query, whose hypergraph matches that of the original $\mathcal{Q}$, and $D_2$ is some database with intervals and $O(|D_2|)$ size. See Figure 1.

**Example 5.1.** Consider the IJ query $Q_3$ from Example 1.1. WLOG let’s take the EJ query $\mathcal{Q}_3$ that results from the reduction:

$$\mathcal{Q}_3 = R_{2,1}(A_1, A_2, B_1) \land S_{2,2}(B_1, B_2, C_1, C_2) \land T_{1,1}(A_1, C_1)$$

Consider an arbitrary database $D = \{D_{R_{2,1}}, D_{S_{2,2}}, D_{T_{1,1}}\}$ over the schema $\{R_{2,1}, S_{2,2}, T_{1,1}\}$. We can reduce solving $\mathcal{Q}_3(D)$ to solving query $Q_3$ over another database $D = \{D_R, D_S, D_T\}$ (whose values are intervals) constructed as follows. Let $F$ be a function that maps binary strings $\{0, 1\}$ into intervals $[x, y]$ for $0 \leq x \leq y \leq 1$, that is defined recursively: $F(\epsilon) = [0, 1]$, $F(\epsilon) = [0, 1/2)$, $F(1) = [1/2, 1)$, $F(0\epsilon) = [0, 1/4)$ and so on. Namely for any given binary string $b$, $F(b \circ o)$ and $F(b \circ \epsilon)$ correspond to the first and second half of $F(b)$ respectively. WLOG we can assume that the domain of $D$ is $[0, 1]^d$, i.e., the set of binary strings of length $d$ for some fixed constant $d$. Construct $D_R$, $D_S$, and $D_T$ as follows:

- $D_R := \{(F(a_1 \circ a_2), F(b_1)) \mid (a_1, a_2, b_1) \in \tilde{D}_{R_{2,1}}\}$,
- $D_S := \{(F(b_1 \circ b_2), F(c_1 \circ c_2)) \mid (b_1, b_2, c_1, c_2) \in \tilde{D}_{S_{2,2}}\}$,
- $D_T := \{(F(a_1), F(c_1)) \mid (a_1, c_1) \in \tilde{D}_{T_{1,1}}\}$.

We can show that $Q_3(D)$ holds if and only if $\mathcal{Q}_3(D)$.

Moreover $|D| = |D|$. This basically proves that solving $Q_3$ is at least as hard as solving $\mathcal{Q}_3$. The same holds for all queries $\{\mathcal{Q}_1, \ldots, \mathcal{Q}_8\}$. In contrast, the forward reduction shows that $Q_3$ is at most as hard as solving the hardest query among $\{\mathcal{Q}_1, \ldots, \mathcal{Q}_8\}$. Together, this implies that $\mathcal{Q}_3$ is exactly as hard as the hardest query among $\{\mathcal{Q}_1, \ldots, \mathcal{Q}_8\}$, meaning that our forward reduction is actually tight. See Appendix C for more details.

**Theorem 5.2.** Let $Q$ be any self-join-free IJ query with hypergraph $H$. Let $\mathcal{Q}$ be any EJ query whose hypergraph is in $r(H)$. For any database $D$, let $\Omega(T(|D|))$ be a lower bound on the time complexity for computing $Q$, where $T$ is a function of the size of the database $D$. There cannot be an algorithm $\mathcal{A}_Q$ that computes $Q(D)$ in time $o(\Omega(T(|D|)))$ (i.e., asymptotically strictly smaller), for any database $D$.

**6 IOTA-ACYCLICITY**

In this section, we answer the following question: Which IJ queries can be computed in linear time (modulo a polylog factor)? To answer this question, we introduce a new notion of acyclicity, called $\iota$-acyclicity, which captures precisely the linear-time computable IJ queries. In other words, $\iota$-acyclicity is for IJ queries what $\alpha$-acyclicity is for EJ queries.

**Definition 6.1 (Iota Acyclic Hypergraph).** A hypergraph $H$ is $\iota$-acyclic if and only if each hypergraph in $r(H)$ is $\alpha$-acyclic.

It is immediate to see why Definition 6.1 defines the hypergraphs of some linear-time computable IJ queries, namely those computed via our reduction. Theorem 6.6 later shows that Definition 6.1 defines in fact all linear-time computable IJ queries.

Since all hypergraphs in $r(H)$ are $\alpha$-acyclic, they correspond to EJ queries that can be computed in linear time [29]. Furthermore, the size of $r(H)$ is independent of the input database and only depends on $H$. Definition 6.1 defines $\iota$-acyclicity indirectly using our reduction. We next show that this is equivalent to a simple syntactic characterisation of the hypergraph of the given IJ query.

**Definition 6.2 (Berge Cycle [10]).** A Berge cycle in $H$ is a sequence $(e^1, e^2, e^3, \ldots, e^n)$ such that: $e^1, \ldots, e^n$ are distinct vertices in $\mathcal{V}$, $e^1, \ldots, e^n$ are distinct hyperedges in $E$, and $e^{n+1} = e^1$, $n \geq 2$; and $e^i$ is in $e^{i+1}$ for each $1 \leq i \leq n$.

**Theorem 6.3 (Iota Acyclicity Characterisation).** A hypergraph is $\iota$-acyclic if and only if it has no Berge cycle of length strictly greater than two.
A natural extension of this work is to refine the acyclicity notion in the presence of both intersection joins and equality joins. This notion necessarily lies between $\alpha$-acyclicity and $\iota$-acyclicity: It is the former when all joins are equality joins, as in the literature, and it is the latter when all joins are intersection joins, as in this paper. A further type of join that is naturally supported by the development in this paper is the membership join: This can be expressed by using a join variable to range over both intervals and points. Our reduction can be optimised to accommodate membership joins, in addition to intersection and equality joins. Characterising the linear-time computable Boolean queries with all three types of joins is an exciting venue of future research.

Acknowledgments

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 682588. AK gratefully acknowledges support from EPSRC via a CASE grant supported by Ordnance Survey.
REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. Foundations of Databases. Addison-Wesley.

[2] Mahmoud Abo Khamis, Ryan R. Curtin, Benjamin Moseley, Hung Q. Ngo, Xuancong Nguyen, Dan Olteanu, and Maximilian Schleicher. 2020. Functional Aggregate Queries with Additive Inequalities. ACM Trans. Databases 35, 4, Article 17 (2020), 41 pages.

[3] Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Attri Rudra. 2016. Joins via Geometric Resolutions: Worst Case and Beyond. ACM Trans. Database Syst. 41, 4 (2016), 22:1–22:45.

[4] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. 2017. What Do Shannon-type Inequalities, Submodular Width, and Disjunctive Datalog Have to Do with One Another? In PODS. 429–444.

[5] Lars Arge, Octavian Procopiciu, Sridhar Ramaswamy, Torsten Suel, and Jeffrey Scott Vitter. 1998. Scalable Sweeping-Based Spatial Join. In VLDB. 570–581.

[6] Panagiotis Bouros and Nikos Mamoulis. 2017. A Forward Scan based Plane Sweep Algorithm for Parallel Interval Joins. Proc. VLDB Endow. 10, 11 (2017), 1346–1357.

[7] Johann Braut-Baron. 2016. Hypergraph Acyclicity Revisited. ACM Comput. Surv. 49, 3 (2016), 54:1–54:26.

[8] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. 1993. Efficient Processing of Spatial Joins Using R-Trees. In SIGMOD. 237–246.

[9] Jost Enderle, Matthias Hannel, and Thomas Seidl. 2004. Joining Interval Data in Relational Databases. In SIGMOD. 683–694.

[10] Ronald Fagin. 1983. Degrees of Acyclicity for Hypergraphs and Relational Database Schemes. J. ACM 30, 3 (1983), 514–550.

[11] Dengfeng Gao, Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. 2005. Join operations in temporal databases. VLDB J. 14, 1 (2005), 2–29.

[12] Xiao-Hu Hu, Ke Yi, and Yuval Taori. 2019. Output-Optimal Massively Parallel Algorithms for Similarity Joins. In SIGMOD. 41–47.

[13] Danila Piatov, Sven Helmer, and Anton Dignös. 2016. An interval join optimized algorithm. In VLDB. 829–840.

[14] Ming-Ling Lo and Chinya V. Ravishankar. 1994. Spatial Joins Using Seeded Trees. ACM Trans. Database Syst. 19, 4 (1994), 529–550.

[15] Nick Koudas and Kenneth C. Sevcik. 1997. Size Separation Spatial Join. In SIGMOD. 237–246.

[16] Mahmoud Abo Khamis, George Chichirim, Antonia Kormpa, and Dan Olteanu. 2021. The Complexity of Boolean Conjunctive Queries with Intersection Joins. Database Syst. 82–94.

[17] Johann Brault-Baron. 2016. Hypergraph Acyclicity Revisited. ACM Comput. Surv. 49, 3 (2016), 54:1–54:26.

[18] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. 1993. Efficient Processing of Spatial Joins Using R-Trees. In SIGMOD. 237–246.

[19] Danila Piatov, Sven Helmer, and Anton Dignös. 2016. An interval join optimized algorithm. In VLDB. 829–840.

[20] Ronald Fagin. 1983. Degrees of Acyclicity for Hypergraphs and Relational Database Schemes. J. ACM 30, 3 (1983), 514–550.

[21] Dengfeng Gao, Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. 2005. Join operations in temporal databases. VLDB J. 14, 1 (2005), 2–29.

[22] Xiao-Hu Hu, Ke Yi, and Yuval Taori. 2019. Output-Optimal Massively Parallel Algorithms for Similarity Joins. In SIGMOD. 41–47.

[23] Danila Piatov, Sven Helmer, and Anton Dignös. 2016. An interval join optimized algorithm. In VLDB. 829–840.
the tuples $t_{a_1}, \ldots, t_{a_k}$ satisfy the equi-join conditions for the variables $X_1, \ldots, X_k$ in the query $Q[X, \sigma]$. Furthermore, all the tuples $t_{a_1}, \ldots, t_{a_k}$ and $t_e$ for each $e \in E \setminus E[X]$ satisfy the rest of the join conditions of the query $\hat{Q}[X, \sigma]$ i.e. the (intersection or equality) joins on variables in $\mathcal{V} \setminus \{X\}$. Hence, $\hat{Q}[X, \sigma](\hat{D}[X, \sigma])$ is true. Therefore, $\hat{Q}[X](D[X])$ is true.

$\Leftarrow$: Assume that $\hat{Q}[X](D[X])$ is true. That means that there exists a permutation $\pi \in \pi(E[X])$ and there exist tuples $t_{a_1} \in \hat{R}(\hat{\pi})$, $\ldots$, $t_{a_k} \in \hat{R}(\hat{\pi})$ and $t_e \in \hat{R}(e)$ for each $e \in E \setminus E[X]$ such that for each $i \in [k]$ we have $\hat{t}_{a_i}(X_i) = \hat{t}_{a_i}(X_i) = \cdots = \hat{t}_{a_i}(X_i)$ and also the tuples satisfy the rest of the join conditions of the query $\hat{Q}[X]$, i.e. the (intersection or equality) joins on variables in $\mathcal{V} \setminus \{X\}$.

By Definition 4.9, for each $i \in [k]$ there exists a tuple $t_{a_i} \in \hat{R}(\hat{\pi}),$ where $\hat{R}(\hat{\pi})$ is a relation over schema $\hat{\sigma} = (\hat{\sigma} \cup \{X_i\}) \setminus \{X_1, \ldots, X_k\}$ in $D$, such that $t_{a_i}(\hat{\sigma}(\hat{\pi})) = \hat{t}_{a_i}(\hat{\sigma}(\hat{\pi}))$ for each $i \in [k]$ and $t_{a_i}(X_i) \cdots \cdot X_2(X_i) \in \hat{C}_{\hat{\pi}}(t_{a_i}(X_i))$ for each $i \in [k]$ and $t_{a_i}(X_i) \cdots \cdot X_2(X_i) = \text{leaf}(t_{a_i}(X_i))$ for $i = k$. Furthermore, by Definition 4.9 we have $\hat{R}(e)$ for each $e \in E$. By Lemma 4.4, the predicate $\langle \forall i \in [k], t_{a_i}(X_i) \rangle \neq \emptyset$ is true. Therefore, the tuples $t_{a_1}, \ldots, t_{a_k}$ satisfy the intersection join condition on variable $X_i$ in query $Q$. Furthermore, all the tuples $t_0, \ldots, t_n$ and $t_e$ for each $e \in E \setminus E[X]$ satisfy the rest of the join conditions of $Q$, i.e. the (intersection or equality) joins on variables in $\mathcal{V} \setminus \{X\}$. Hence $Q(D)$ is true. □

C MISSING DETAILS FROM SECTION 5

THEOREM 5.2 Let $Q$ be any self-join-free EJ query with hypergraph $H$. Let $\hat{Q}$ be any EJ query whose hypergraph is in $\tau(H)$. For any database $D$, let $\Omega(T(\hat{D}))$ be a lower bound on the time complexity for computing $\hat{Q}$, where $T$ is a function of the size of the database $D$. There cannot be an algorithm $A_Q$ that computes $Q(D)$ in time $o(T(\hat{D}))$ (i.e., asymptotically strictly smaller), for any database $D$.

Proof. Suppose, for a contradiction, that there is such an algorithm $A_Q$. We will show that we can construct an algorithm $A_Q'$ based on $A_Q$ that can solve $\hat{Q}(\hat{D})$ in time complexity $o(T(\hat{D}))$ (i.e., asymptotically strictly smaller), for any input database $\hat{D}$ of $\hat{Q}$.

Let $\hat{D}$ be any input database for EJ query $Q$. We will base our construction on the structure of the segment tree. Let $brep(x)$ be the binary representation of the natural number $x$. WLOG we can assume that each value in $\hat{D}$ is a binary string of length exactly $b$ for some constant $b$. Let $n = |E[\hat{D}]| = |E|.$

Consider a slightly modified version of a perfect binary tree with $2^n - b$ leaves (so with height equal to $n - b$) where, for each node $u$, we have $\text{seg}(u) := |x, y|$, where $x$ and $y$ are natural numbers such that $\text{brep}(x) := \langle 1 \rangle$ $u \in \text{seg}(x)$ and $\text{brep}(y) := \langle 0 \rangle$ $u \in \text{seg}(y)$, where $t \in \cdot b - a \cdot |a \cdot t$ represents the string $\langle 0 \rangle$ repeated $t$ times (and the same for $\langle 1 \rangle$). All the properties of a segment tree that are relevant for this proof hold for this version too.

Similar to the one-step forward reduction from Section 4.2, we define a one-step backward reduction and then apply it repeatedly.

DEFINITION 6.1 (ONE-STEP BACKWARD DATABASE TRANSFORMATION). Given an EJ $Q$, let $[X]$ be an interval variable of $Q$ and let $\sigma \in \pi(E[X])$. Let $\hat{Q}(X, \sigma)$ be the EJ resulting from the one-step query rewriting from Definition 4.7. Let $D[X, \sigma]$ be an arbitrary database instance over the schema of $\hat{Q}(X, \sigma)$. We construct a database instance $\hat{D}$ over the schema of $Q$ as follows. For each tuple $t \in \hat{R}(\hat{\pi})$, we construct a tuple $t \in R(\pi)$ such that:

- $t(\hat{\pi}(\hat{\pi})) = \hat{t}(\hat{\pi}(\hat{\pi})) \setminus \{X_1, \ldots, X_k\})$ for each $i \in [k]$. Let $k := |E[X]|$ be the number of hyperedges that contain each $u$. Each vertex $u$ that occurs in $n_u$ hyperedges in $H$ corresponds to $n_u$ vertices in $\hat{H}$ denoted by $\tilde{u}_1, \ldots, \tilde{u}_{n_u}$.
DEFINITION D.1. Consider a hypergraph \( \mathcal{H} = (V, E) \). Let \( \tilde{\mathcal{H}} = (\tilde{V}, \tilde{E}) \) be any member of \( r(\mathcal{H}) \). (1) Let \( v_{\tilde{\mathcal{H}}} : \tilde{V} \rightarrow V \) be the surjective function that maps each vertex \( \tilde{u} \in \tilde{V} \) to the corresponding vertex \( u \in V \). (2) Let \( e_{\tilde{\mathcal{H}}} : \tilde{E} \rightarrow E \) be the bijective function that maps each hyperedge \( \tilde{e} \in \tilde{E} \) to the corresponding hyperedge \( e = \{ u \mid \tilde{u} \in \tilde{e} \} \in E \). Whenever it is clear from the context, we omit the subscript from the names of the functions.

PROPERTY D.2 (PROPERTIES OF I - TO - J REDUCTION). Consider a hypergraph \( \mathcal{H} = (V, E) \). Let \( \mathcal{H}' = (\tilde{V}, \tilde{E}) \) be any member of \( r(\mathcal{H}) \).

(1) For each hyperedge \( \tilde{e} \in \tilde{E} \) and each vertex \( \tilde{u} \in \tilde{V} \), if \( \tilde{u} \in \tilde{e} \) in \( \tilde{\mathcal{H}} \) then \( v(\tilde{u}) \in e(\tilde{e}) \) in \( \mathcal{H} \).

(2) For each hyperedge \( \tilde{e} \in \tilde{E} \) and each vertex \( u \in V \), \( \tilde{u} \in \tilde{e} \) in \( \tilde{\mathcal{H}} \) if and only if \( v(\tilde{u}) \in e(\tilde{e}) \) in \( \mathcal{H} \).

(3) For any two vertices \( \tilde{u}_i, \tilde{u}_j \in \tilde{V} \) with \( i < j \) and \( \tilde{e} \in \tilde{E} \), \( \tilde{u}_i \in \tilde{e} \) if and only if \( v(\tilde{u}_i) \in e(\tilde{e}) \) in \( \mathcal{H} \).

LEMMA D.3. If \( \tilde{\mathcal{H}} \) has a Berge cycle \( (\tilde{e}_1, \tilde{e}_2, \tilde{e}_3, \ldots, \tilde{e}_k, \tilde{e}_{k+1} = \tilde{e}_1) \) of length \( k \) such that \( v(\tilde{e}_1), \ldots, v(\tilde{e}_k) \) are pairwise distinct vertices from \( \tilde{V} \), then \( \mathcal{H} \) also has a Berge cycle of length \( k \).

THEOREM 6.3 A hypergraph \( \mathcal{H} = (V, E) \) is \( i \)-acyclic if and only if \( \tilde{\mathcal{H}} \) has no Berge cycle of length strictly greater than two.

PROOF. \( \Rightarrow \) Assume for a contradiction that \( \mathcal{H} \) has a Berge cycle of length strictly greater than two, or equivalently at least three. Hence, there exist a cyclic sequence \( (\tilde{e}_1, \tilde{e}_2, \tilde{e}_3, \ldots, \tilde{e}_k, \tilde{e}_k+1 = \tilde{e}_1) \) such that \( k \geq 3 \), \( \tilde{e}_i, \ldots, \tilde{e}_k \) are pairwise distinct vertices from \( \tilde{V} \), \( \tilde{e}_1, \ldots, \tilde{e}_k \) are pairwise distinct hyperedges from \( \tilde{E} \), and for each \( 1 \leq i \leq k \), we have \( \tilde{e}_i \subseteq \tilde{e}_{i+1} \) by Claim D.4, \( \tilde{e}_{i+1} \subseteq \tilde{e}_i \). By our construction in Algorithm 1, there exists a hypergraph \( \mathcal{H}' = (\tilde{V}', \tilde{E}') \in r(\mathcal{H}) \) such that for each \( 1 \leq i \leq k \) we have \( \tilde{e}_i, \ldots, \tilde{e}_k \) are distinct hyperedges in \( \tilde{E} \) of pairwise distinct vertices from \( \tilde{V} \).

\( \mathcal{H}' \) has the following three properties:

\( \bullet \) For each \( 1 \leq i \leq k \) the vertex \( \tilde{v}_{\tilde{e}_{i+1}} \) belongs to precisely two hyperedges from \( \tilde{E} \).

\( \bullet \) For each \( 1 \leq i \leq k \) with \( i \neq j \), the hyperedge \( e_{\tilde{e}_{i+1}} \) is not contained in hyperedge \( e_{\tilde{e}_j} \) by (1) because the vertex \( \tilde{v}_{\tilde{e}_{i+1}} \not\in e_{\tilde{e}_j} \).

\( \bullet \) For each \( 1 \leq i \leq k \), the hyperedge \( e_{\tilde{e}_i} \) is not contained in hyperedge \( e_{\tilde{e}_{i+1}} \) by (2).

Let \( \tilde{V}' = \{ \tilde{v}_{\tilde{e}_{i+1}} \mid 1 \leq i \leq k \} \) and \( \tilde{E}' = \{ e_{\tilde{e}_{i+1}} \mid 1 \leq i \leq k \} \). Note that \( \tilde{V}' \subseteq \tilde{V} \) and \( \tilde{E}' \subseteq \tilde{E} \). Therefore, we can prove the following:

\( \bullet \) For each \( 1 \leq i \leq k \), the vertex \( \tilde{v}_{\tilde{e}_{i+1}} \) belongs to \( e_{\tilde{e}_i} \).

\( \bullet \) For each \( 1 \leq i < k \), the vertex \( \tilde{v}_{\tilde{e}_{i+1}} \) belongs to \( e_{\tilde{e}_i} \) but not to \( e_{\tilde{e}_{i+1}} \).

\( \bullet \) For each \( 1 \leq i < k \), the vertex \( \tilde{v}_{\tilde{e}_{i+1}} \) belongs to \( e_{\tilde{e}_i} \) but not to \( e_{\tilde{e}_{i+1}} \).

Therefore, \( \tilde{\mathcal{H}} \) is not \( i \)-acyclic.
vertex $\tilde{v}$ as well. The reverse case is analogous due to symmetry. Contradiction.

**Proof of Claim D.5.** The proof is similar to the proof of Claim D.4

**E SOLVING THE TRIANGLE IQ USING FAQ-AI**

Consider the triangle intersection join query $Q_\Delta$, from Section 1.1. While we showed in Section 1.1 that our approach solves this query written as follows:

$$\text{The same applies to the other two interval variables \(A_1, A_2\) as well, and \(A_3\) in relation \(R\). Similarly, we define \(A_1(T)\) and \(A_2(T)\). For the intervals \([A_1(T), A_2(T)]\) and \([A_1(T), A_3(T)]\) to overlap, the following condition must hold:}

$$\left( A_1(T) \leq A_1(R) \leq A_3(T) \right) \lor \left( A_3(T) \leq A_3(R) \leq A_1(T) \right) \quad (3)$$

For each variable $X \in \{A_1, A_2, A_3\}$, let $F(X)$ denote the set of relations containing $X$, i.e. $F(A_i) = \{T, R\}$, $F(B) = \{S, T\}$, and $F(C) = \{S, T\}$. Note that (3) can be written equivalently as:

$$\bigvee_{W \in F(A)} A_1(W) \leq A_1(V) \leq A_3(W) \quad (4)$$

The same applies to the other two interval variables $[B]$ and $[C]$. Let:

$$\mathcal{R} := \langle A_1(R), A_2(R), B_1(R), B_3(R) \rangle,$$

$$\mathcal{S} := S(B_1(S), B_2(S), C_1(S), C_2(S)),$$

$$\mathcal{T} := T(A_1(T), A_2(T), C_1(T), C_2(T)),$$

and

$$\mathcal{A} := \langle A_1(W_1), A_2(W_1) \rangle,$$

$$\mathcal{B} := B_1(W_1) \geq B_2(W_1),$$

$$\mathcal{C} := C_1(W_1) \leq C_2(W_1).$$

After distributing disjunctions over conjunctions, $Q_\Delta$ can be written as follows:

$$Q_\Delta = \bigvee_{(V_A, V_B, V_C) \in F(A) \times F(B) \times F(C)} \mathcal{R} \land \mathcal{S} \land \mathcal{T} \land \mathcal{A} \land \mathcal{B} \land \mathcal{C}. \quad (5)$$

For each $(V_A, V_B, V_C) \in F(A) \times F(B) \times F(C)$, the inner conjunction in (5) is an FAQ-AI query [2]. While solving each such query, it is possible to relax the definition of tree decompositions thus extending the set of valid tree decompositions and potentially reducing the fractional hypertree and submodular widths, ultimately resulting in the relaxed versions of these widths $\text{fhtw}_\ell$ and $\text{subw}_\ell$ respectively [2]. In particular, in a relaxed tree decomposition, we no longer require each inequality to have its variables contained in one bag of the tree. Instead, it suffices to have its variables contained in two adjacent bags in the tree.

Fix an arbitrary $(V_A, V_B, V_C) \in F(A) \times F(B) \times F(C)$ and let $Q$ be the resulting FAQ-AI query corresponding to the inner conjunction in (5). Note that for every pair of the relations $R, S$ and $T$, the query $Q$ contains at least one inequality between two variables from that pair. Hence if we distribute the relations $R, S$ and $T$ among the three or more bags, there will be an inequality between two non-adjacent bags thus violating the condition for a relaxed tree decomposition. Therefore, every relaxed tree decomposition of $Q$ must have at most two bags where each one the relations $R, S$ and $T$ falls within one bag. Consequently, there will be one bag with (at least) two relations. Noting that the variables of relations $R, S$ and $T$ are pairwise disjoint, this implies that $\text{fhtw}_\ell(Q) \geq 2$. To minimize $\text{fhtw}_\ell(Q)$, an optimal tree decomposition would have two bags with two relations in one bag and the third relation in the other, resulting in $\text{fhtw}_\ell(Q) = 2$.

The relaxed submodular width $\text{subw}_\ell(Q)$ is not any better in this case. Consider the following function $\tilde{h} : 2^{\text{vars}(Q)} \to \mathbb{R}^+$:

$$\tilde{h}(X) := \left| \frac{X}{4} \right|, \forall X \subseteq \text{vars}(Q). \quad (6)$$

Recall notation from Section A and [2, 21]. The above $\tilde{h}$ is a modular function hence it is submodular. Since it is also monotone, $\tilde{h}$ is a polymatroid, i.e. $\tilde{h} \in \Gamma_{\text{vars}(Q)}$, where $\Gamma_{\text{vars}(Q)}$ denotes the set of polymatroids over the variables $\text{vars}(Q)$ [2]. Moreover for each finite input relation $E \in \{R, S, T\}$, we have $\tilde{h}(E) = 1$ since each one of these relations has four variables, i.e. $|E| = 4$. Therefore $\tilde{h}$ is edge dominated, i.e. $\tilde{h} \in \mathcal{ED}(Q)$ where $\mathcal{ED}(Q)$ denotes the set of edge dominated functions $h : 2^{\text{vars}(Q)} \to \mathbb{R}^+$. Recall the definition of $\text{subw}_\ell(Q)$ for an FAQ-AI query $Q$ from [2] where $\mathcal{TD}_\ell(Q)$ denotes the set of relaxed tree decompositions of $Q$:

$$\text{subw}_\ell(Q) := \max_{h \in \mathcal{ED}(Q) \cap \Gamma_{\text{vars}(Q)}} \min_{(T, \chi) \in \mathcal{TD}_\ell(Q)} \max_{t \in \mathcal{V}(T)} h(\chi(t)). \quad (7)$$

Based on the above definition and by choosing $\tilde{h} \in \mathcal{ED}(Q) \cap \Gamma_{\text{vars}(Q)}$, we have

$$\text{subw}_\ell(Q) \geq \min_{(T, \chi) \in \mathcal{TD}_\ell(Q)} \max_{t \in \mathcal{V}(T)} \tilde{h}(\chi(t)).$$

However for each relaxed tree decomposition $(T, \chi) \in \mathcal{TD}_\ell(Q)$, we argued before that there must exist some bag $t^* \in V(T)$ containing at least two of the input relations $\{R, S, T\}$ hence at least 2 distinct variables, meaning that $|\chi(t^*)| \geq 8$. From (6), we have $\tilde{h}(\chi(t^*)) \geq 2$ which implies that $\text{subw}_\ell(Q) \geq 2$. And since $\text{subw}_\ell(Q) \leq \text{fhtw}_\ell(Q)$ for any query $Q$ according to [2], we have

$$\text{subw}_\ell(Q) = \text{fhtw}_\ell(Q) = 2. \quad (8)$$

Finally according to Theorem 3.5 in [2], the time complexity in FAQ-AI involves an extra factor of $(\log N)^{\max(k-1,1)}$ where $k$ is the number of inequalities that involve variables from two adjacent bags (i.e. that are not contained in a single bag) in an optimal relaxed tree decomposition. In query (5), when constructing any optimal relaxed tree decomposition involving two relations in one bag (say $R$ and $S$) and the third relation in another bag, there will be exactly 4 inequalities involving variables from both bags. Hence $k = 4$ and the overall time complexity of FAQ-AI for $Q_\Delta$ is $O(N^4 \log^3 N)$. 

