Factorised Representations of Query Results: Size Bounds and Readability

Dan Olteanu and Jakub Závodný
Department of Computer Science
University of Oxford
{dan.olteanu,jakub.zavodny}@cs.ox.ac.uk

ABSTRACT

We introduce a representation system for relational data based on algebraic factorisation using distributivity of product over union and commutativity of product and union.

We give two characterisations of conjunctive queries based on factorisations of their results whose nesting structure is defined by so-called factorisation trees.

The first characterisation concerns sizes of factorised representations. For any query, we derive a size bound that is asymptotically tight within our class of factorisations.

We also characterise the queries by tight bounds on the readability of the provenance of result tuples and define syntactically the class of queries with bounded readability.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Relational Databases, Query Processing*

General Terms

Theory, Algorithm Design

Keywords

conjunctive queries, data factorisation, hierarchical queries, provenance, query evaluation, readability, size bounds

1. INTRODUCTION

This paper studies two properties related to the representation of results of conjunctive queries: size and readability. In this study, we depart from the standard flat representation of query results as sets of tuples and consider instead *factorised* representations of query results. The relationship between a flat representation and an equivalent factorised representation is on a par with the relationship between logic functions in disjunctive normal form and their equivalent nested forms obtained by algebraic factorisation. This work lies at the foundation of a new kind of database systems that present relations at the logical layer and use equivalent factorised representations at the physical layer. The underlying observation is that factorised representations can boost the performance of query processing in relational databases in case of large input, intermediate, or final results, since they can be exponentially more succinct than flat representations. This paper characterises conjunctive queries by their ability to map flat input data to small factorised results; the case of factorised input data is deferred to future work.

We derive a number of results about factorised representations of query results and provenance. The structure of such representations is defined by so-called factorisation trees, which can be derived from the query hypergraph.

The size of a factorised representation is the number of its data values. For any (non-Boolean) conjunctive query Q, there is a rational number $s(Q)$ such that (1) for any database $\mathbf{D}$, there exists a factorised representation of $Q(\mathbf{D})$ with size at most $|\mathcal{S}| \cdot |\mathbf{D}|^{s(Q)}$, where $|\mathcal{S}|$ is the size of the result schema, and (2) for any factorisation tree $\mathcal{T}$ of Q, there exist arbitrarily large databases $\mathbf{D}$ for which the factorised representation of $Q(\mathbf{D})$ over $\mathcal{T}$ has size at least $(|\mathbf{D}|/|Q|)^{s(Q)}$. The results of Boolean queries either consist of the nullary tuple ($\langle\rangle$) or are the empty relation ($\emptyset$), both of size 1.

The parameter $s(Q)$ is the fractional edge cover number of a subquery of Q. The problem of deciding whether $s(Q) \leq k$ for a given query Q and rational number k is in NP. However, its exact complexity remains open. A factorised representation of the query result can be computed in time $O(|\mathbf{D}|^{s(\mathcal{T})})$, where $\mathcal{T}$ is a factorisation tree of Q without projections and extends a factorisation tree of Q.

We also characterise conjunctive queries by the readability of provenance of their result tuples. A factorisation Φ of a relation or Boolean function is *read-k* if each value or variable appears at most k times in Φ. The readability of a relation or function Φ is then the smallest number k such that there is a read-k factorisation of Φ [5, 4, 3].

For any conjunctive query Q, there is a rational number $r(Q)$ called *readability width* such that for any database $\mathbf{D}$, the readability of the provenance of any tuple in the result $Q(\mathbf{D})$ is at most read-$M \cdot |\mathbf{D}|^{r(Q)}$, where M is the maximum number of repeating relation symbols in Q. This upper bound is asymptotically matched by a lower bound of read-$(|\mathbf{D}|/|Q|)^{r(Q)}$ for arbitrarily large databases $\mathbf{D}$ and any factorisation tree within a special class.

A remarkable dichotomy holds for readability: *hierarchical queries* [2] are precisely those conjunctive queries with readability width 0, and the readability width of any non-hierarchical query is at least 1. Moreover, for non-repeating non-hierarchical queries there exist arbitrarily large databases $\mathbf{D}$ such that the readability of the provenance of some result tuple is $\Omega(\sqrt{|\mathbf{D}|})$. The hierarchical property thus characterises queries with bounded readability and the readability width characterises how "far" a query is from being hierarchical. Hierarchical queries are intimately connected to tractability in probabilistic databases [20], admit parallel evaluation with one broadcast step [13], and have one-step streaming evaluation in the finite cursor model [9].

2. PRELIMINARIES

Databases. We consider relational databases with named attributes. An attribute A is any symbol. A schema $\mathcal{S}$ is a set of attributes and a tuple t of schema $\mathcal{S}$ is a mapping from $\mathcal{S}$ to a domain $\mathcal{D}$. A relation $\mathbf{R}$ over $\mathcal{S}$ is a set of tuples of schema $\mathcal{S}$. A database $\mathbf{D}$ is a collection of relations. The size $|\mathbf{R}|$ of $\mathbf{R}$ is the number of its tuples; the size $|\mathbf{D}|$ of a database $\mathbf{D}$ is the sum of the sizes of its relations.

Queries. In this paper we consider conjunctive queries written in the canonical relational algebra form

$$\pi_{\mathcal{P}}(\sigma_{\psi}(R_1 \times \ldots \times R_n)),$$

where $R_1, \ldots, R_n$ are distinct relation symbols over disjoint schemas $\mathcal{S}_1, \ldots, \mathcal{S}_n$, ψ is a conjunction of equalities of the form $A_1 = A_2$ with attributes A_1 and A_2, and the projection list $\mathcal{P} \subseteq \bigcup_i \mathcal{S}_i$. The attributes in $\mathcal{P}$ are called the *head* attributes. If $\mathcal{P} = \bigcup_i \mathcal{S}_i$ we can drop the projection $\pi_{\mathcal{P}}$ and Q is an equi-join query. The equi-join of Q is the query $\hat{Q} = \sigma_{\psi}(R_1 \times \ldots \times R_n)$. The *size* of Q is $|Q| = n$.

Although we require for simplicity that the relation symbols are distinct, they may be mapped to the same relation. Our query language can thus capture queries with self-joins, but we assume that different copies of relations are already given different names and we thus need not use explicit aliases in the query syntax. Similarly, the attribute names are assumed distinct and we thus need not use explicit renaming operators in the query syntax.

For an attribute A in a query Q, we denote by A^* its equivalence class, i.e., the set consisting of A and of all attributes that are transitively equal to A in the selection condition of Q. We denote by rel(A) the set of all relation symbols of Q with attributes in A^*. The hypergraph of Q has one node for each attribute class of Q and for each relation symbol R has a hyperedge over all nodes with attributes of R.

Definition 1. [2] A conjunctive query is *hierarchical*, if for any two non-head attributes A and B, either rel$(A) \subseteq$ rel(B), or rel$(A) \supset$ rel(B), or rel$(A) \cap$ rel$(B) = \emptyset$. $\square$

Computational model. We use the uniform-cost RAM model where the values of the domain $\mathcal{D}$ as well as the pointers into the database are of constant size.

3. FACTORISED REPRESENTATIONS

In this section we introduce the notion of factorised representations of relations. Such representations are expressed in a fragment of relational algebra consisting of unions, Cartesian products, and singleton relations.

Definition 2. Let $\mathcal{S}$ be a relational schema. A *factorised representation*, or *f-representation* for short, over $\mathcal{S}$, is a relational algebra expression of the form

- $\emptyset$, the empty relation over schema $\mathcal{S}$,
- $\langle\rangle$, the relation consisting of the nullary tuple, if $\mathcal{S} = \emptyset$,
- $\langle A\!:\!a\rangle$, the unary relation with a single tuple with value a, if $\mathcal{S} = \{A\}$ and a is a value in the domain $\mathcal{D}$,
- (E), where E is an f-representation over $\mathcal{S}$,
- $E_1 \cup \cdots \cup E_n$, where each E_i is an f-representation over $\mathcal{S}$,
- $E_1 \times \cdots \times E_n$, where each E_i is an f-representation over $\mathcal{S}_i$ and $\mathcal{S}$ is the disjoint union of all $\mathcal{S}_i$. $\square$

The expressions $\langle A\!:\!a\rangle$ are called *singletons of type* A or A-singletons and the expression $\langle\rangle$ is called the nullary singleton. The *size* $|E|$ of an f-representation E is the number of singletons plus the number of empty relations in E.

Any f-representation E can be interpreted as a relation $[\![E]\!]$. We say that $[\![E]\!]$ is the relation of E. Different f-representations can represent the same relation. Two f-representations E_1 and E_2 are *equivalent* if $[\![E_1]\!] = [\![E_2]\!]$.

Any relation has a so-called *flat* f-representation that is a (possibly empty) union of products of singletons, where each product of singletons represents one tuple in the relation. This property of f-representations is called completeness.

Proposition 1. *Factorised representations form a complete representation system for relational data.*

Although any relation has a flat f-representation, nested f-representations can be exponentially more succinct than their equivalent flat f-representations, where the exponent is the size of the schema.

Example 1. The f-representation $(\langle A_1\!:\!0\rangle \cup \langle A_1\!:\!1\rangle) \times \ldots \times (\langle A_n\!:\!0\rangle \cup \langle A_n\!:\!1\rangle)$ has size $2n$, while any equivalent flat f-representation has size 2^{n+1}. $\square$

When describing algorithms on f-representations, we assume wlog that f-representations are given as parse trees in memory. We also assume that they are *normalised* in the sense that no f-representation contains an empty relation or a nullary singleton, unless it is itself the empty relation or the nullary singleton. An f-representation can be normalised in linear time similarly to simplification by propagation of the constants *true* and *false* in Boolean functions. By expanding an f-representation using the distributivity of product over union, we obtain an equivalent flat f-representation that is a union of products of singletons (akin to disjunctive normal form of a Boolean function). These products can be enumerated efficiently.

Proposition 2. *The products of singletons in an f-representation E over a schema $\mathcal{S}$ can be enumerated with $O(|E|)$ space and precomputation time and $O(|\mathcal{S}|)$ delay.*

The products of singletons obtained by expanding an f-representation E using the distributivity of product over union correspond to the tuples of $[\![E]\!]$ under bag semantics. The present work is concerned with set semantics and in the sequel we thus only consider f-representations with distinct products of singletons. This implies that for any such f-representation the distinct tuples of the represented relation $[\![E]\!]$ can be enumerated efficiently *in some order*. The problem of tuple enumeration under a given order is out of the scope of this paper. Allowing duplicate tuples can make f-representations more succinct, yet the enumeration of distinct tuples may require larger delay or precomputation time.

Remark 1. Earlier work in incomplete databases has introduced a representation system called world-set decompositions [16] to represent succinctly sets of possible worlds. Such decompositions can be seen as f-representations whose structure is a product of unions of products of singletons. $\square$

4. F-TREES FOR RELATIONS

We next introduce factorisation trees that define the schemas as well as the nesting structures of f-representations.

Definition 3. A *factorisation tree*, or f-tree for short, over a possibly empty schema $\mathcal{S}$ is an unordered rooted forest with each node labelled by a non-empty subset of $\mathcal{S}$ such that each attribute of $\mathcal{S}$ occurs in exactly one node. $\square$

We next define f-representations over a given f-tree.

Definition 4. An f-representation E over a given f-tree $\mathcal{T}$ is recursively defined as follows:

- If $\mathcal{T}$ is a forest of trees $\mathcal{T}_1, \dots, \mathcal{T}_k$, then
$$E = E_1 \times \cdots \times E_k$$
where each E_i is an f-representation over $\mathcal{T}_i$.
- If $\mathcal{T}$ is a single tree with a root labelled by $\{A_1, \dots, A_k\}$ and a non-empty forest $\mathcal{U}$ of children, then
$$E = \bigcup\nolimits_a \langle A_1 : a\rangle \times \cdots \times \langle A_k : a\rangle \times E_a$$
where each E_a is an f-representation over $\mathcal{U}$ and the union $\bigcup_a$ is over a collection of distinct values a.
- If $\mathcal{T}$ is a single node labelled by $\{A_1, \dots, A_k\}$, then
$$E = \bigcup\nolimits_a \langle A_1 : a\rangle \times \cdots \times \langle A_k : a\rangle.$$
- If $\mathcal{T}$ is empty, then $E = \emptyset$ or $E = \langle\rangle$. $\square$

In an f-representation over an f-tree $\mathcal{T}$, all attributes labelling a node in $\mathcal{T}$ have equal values in the represented relation. The shape of $\mathcal{T}$ defines a hierarchy of attributes by which we group the represented relation in the factorisation. We group its tuples by the values of the attributes labelling the root, factor out the common values in each group, and then continue recursively on each group using the attributes lower in the f-tree. Branching into several subtrees denotes a product of f-representations over the individual subtrees.

Example 2. Consider a relation over schema $\{A, B, C\}$ and domain $\mathcal{D} = \{1, \dots, 5\}$ that represents the inequalities $A < B < C$. An f-representation of this relation is

$$\begin{array}{ll}
\langle B:2\rangle \times \langle A:1\rangle & \times\, (\langle C:3\rangle \cup \langle C:4\rangle \cup \langle C:5\rangle) \cup \\
\langle B:3\rangle \times (\langle A:1\rangle \cup \langle A:2\rangle) & \times\, (\langle C:4\rangle \cup \langle C:5\rangle) \cup \\
\langle B:4\rangle \times (\langle A:1\rangle \cup \langle A:2\rangle \cup \langle A:3\rangle) & \times\, \langle C:5\rangle.
\end{array}$$

over the f-tree
$$\begin{tikzcd} & B \arrow[dl, dash] \arrow[dr, dash] & \\ A & & C \end{tikzcd}$$
$\square$

For a given f-tree $\mathcal{T}$ over a schema $\mathcal{S}$, not all relations over $\mathcal{S}$ have an f-representation over $\mathcal{T}$ since the subexpressions over subtrees that are siblings in $\mathcal{T}$ must appear in a product and this may not be possible for all relations.

Example 3. The relation $\{\langle 1, 1, 1\rangle, \langle 2, 1, 2\rangle\}$ over schema $\{A, B, C\}$ does not admit an f-representation over the f-tree from Example 2, since any such f-representation must essentially be of the form $\langle B:1\rangle \times E_A \times E_C$, where E_A is a union of A-values and E_C is a union of C-values. $\square$

Lemma 1. *Let* $\mathbf{R}$ *be a relation over schema* $\mathcal{S}$ *and* $\mathcal{T}$ *be an f-tree over* $\mathcal{S}$*. If there is an f-representation of* $\mathbf{R}$ *over* $\mathcal{T}$*, then it is unique up to commutativity of union and product.*

If it exists, the f-representation of relation $\mathbf{R}$ over the f-tree $\mathcal{T}$ is denoted by $\mathcal{T}(\mathbf{R})$ and can be constructed as follows:

- If $\mathcal{T}$ is a forest $\mathcal{T}_1, \dots, \mathcal{T}_k$ then
$$\mathcal{T}(\mathbf{R}) = \mathcal{T}_1(\pi_{\mathcal{T}_1}(\mathbf{R})) \times \cdots \times \mathcal{T}_k(\pi_{\mathcal{T}_k}(\mathbf{R})).$$
- If $\mathcal{T}$ is a tree with root $\{A_1, \dots, A_k\}$ and a non-empty forest $\mathcal{U}$ of children, then
$$\mathcal{T}(\mathbf{R}) = \bigcup\nolimits_{a \in A} \langle A_1 : a\rangle \times \cdots \times \langle A_k : a\rangle \times \mathcal{U}(\pi_{\mathcal{U}} \sigma_{\mathcal{A}=a} \mathbf{R}).$$
- If $\mathcal{T}$ is a node $\{A_1, \dots, A_k\}$ then
$$E = \bigcup\nolimits_{a \in A} \langle A_1 : a\rangle \times \cdots \times \langle A_k : a\rangle.$$
- If $\mathcal{T}$ is empty, then $E = \emptyset$ if $\mathbf{R}$ is empty and $E = \langle\rangle$ if $\mathbf{R}$ consists of the nullary tuple.

In the above equations, we considered $A = \pi_{A_1}(\mathbf{R}) = \cdots = \pi_{A_k}(\mathbf{R})$, $(\mathcal{A} = a)$ denotes $\bigwedge_i (A_i = a)$, and $\pi_{\mathcal{U}}$ denotes the projection on the attributes of the f-tree $\mathcal{U}$.

The above construction ensures that $\mathcal{T}(\mathbf{R})$ is a normalised f-representation and encodes distinct products of singletons that can be enumerated according to Proposition 2.

We next define two notions that are necessary in subsequent sections to characterise sizes of f-representations over f-trees. For an attribute A in an f-tree $\mathcal{T}$, we denote by anc(A) the set of all attributes at nodes that are ancestors of the node labelled by A in $\mathcal{T}$, and by path(A) the set of attributes at the ancestors of the node of A *and* at the node of A; thus, path$(A) = \text{anc}(A) \cup A^*$.

Example 4. In the left f-tree in Figure 1, path(C) is the union of all attribute sets at nodes on the root-to-leaf path ending at C: path$(C) = \{A_R, A_S, A_T, B_R, B_S, C\}$. $\square$

5. F-TREES FOR QUERY RESULTS

In this section, we characterise the f-trees over which the result of a conjunctive query Q is factorisable for *any* input database. These f-trees can be inferred from the query Q. We first discuss the simpler case of equi-join queries, i.e., queries without projection, and then show how to extend it to arbitrary conjunctive queries.

In the following statements, we only consider f-trees whose nodes are bijectively labelled by the equivalence classes of head attributes in the input query Q. This assumption is justified later in the section.

Proposition 3. *Given an equi-join query* Q*,* $Q(\mathbf{D})$ *has an f-representation over an f-tree* $\mathcal{T}$ *for any database* $\mathbf{D}$ *iff for each relation in* Q *its attributes lie along a root-to-leaf path in* $\mathcal{T}$*.*

The condition in Proposition 3 is called the *path condition.* Any f-tree satisfying Proposition 3 is *valid* for the query Q, or simply an f-tree of Q.

Example 5. Consider the relations R, S and T over schemas $\{A_R, B_R, C\}$, $\{A_S, B_S, D\}$ and $\{A_T, E_T\}$ respectively and the query $Q_1 = \sigma_\varphi(R \times S \times T)$ with $\varphi = (A_R = A_S = A_T, B_R = B_S)$. The left and middle f-trees in Figure 1 are valid for Q_1. The right f-tree is invalid since the attributes A_S and D of S are not on a common root-to-leaf path. $\square$

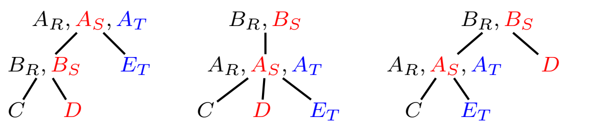

Figure 1: Left to right: two valid f-trees $\mathcal{T}_1$ and $\mathcal{T}_2$ and one invalid f-tree for the query Q_1 in Example 5.

The intuition behind Proposition 3 is as follows. The path condition states that the attributes of a relation are in general "dependent" on each other: as shown in Example 3, the set of combinations of values of two attributes of a relation may not be expressible as the product of two unions of values. Dependent attributes thus need to be along the same path in an f-tree $\mathcal{T}$; if they would be on different paths in $\mathcal{T}$, then their values appear in two sub-expressions that are in a product in any f-representation over $\mathcal{T}$. In case the path condition is satisfied, we can produce an f-representation of $Q(\mathbf{D})$ over $\mathcal{T}$ by first joining the input relations on the attributes in the root of $\mathcal{T}$ and then on the attributes in the lower nodes, leaving products unexpanded whenever $\mathcal{T}$ branches into independent subtrees.

Proposition 3 does not capture all f-trees that can be derived from queries: it only deals with f-trees whose nodes are labelled by the attribute classes of Q. For an f-tree to factorise the results of Q, only attributes within the same class may appear at a node, yet nodes need not be labelled by whole attribute classes. The f-trees left out are those where several nodes have attributes from the same class, and for each class, among the nodes containing attributes of that class, there is one which is an ancestor of the others. For any such f-tree, the f-tree constructed by pushing up all attributes of a class to the top-most node labelled by an attribute in that class defines f-representations with smaller or equal size and readability. For the purpose of this paper, Proposition 3 thus characterises all interesting f-trees.

We next extend the above characterisation from equi-join to arbitrary conjunctive queries. The difference is that projections can introduce additional dependencies between head attributes: if we join two relations on attributes that are subsequently projected away, the remaining attributes from the two relations become dependent as enforced by the join. These dependent attributes thus need to lie along the same path in the f-tree. To capture this observation, we lift dependencies from attributes to nodes in f-trees.

Definition 5. Let Q be a conjunctive query.

Two relations R and S in Q are *dependent* if

- They have attributes in a common class that does not contain head attributes of Q, or
- There exists a relation T in Q such that R and T are dependent and S and T are dependent.

Two attributes are *dependent* if they belong to the same relation or to dependent relations.

Two nodes are *dependent* if their attributes belong to equivalence classes of dependent attributes. $\square$

For conjunctive queries, Proposition 3 extends as follows.

PROPOSITION 4. *Given a conjunctive query Q, $Q(\mathbf{D})$ has an f-representation over an f-tree $\mathcal{T}$ for any database $\mathbf{D}$ iff any two dependent nodes lie along a root-to-leaf path in $\mathcal{T}$.*

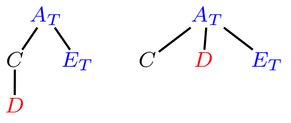

Figure 2: Left to right: a valid and an invalid f-tree for the query $\pi_{A_T,C,D,E_T} Q_1$ in Example 6.

Similar to Proposition 3, the condition in Proposition 4 is called the path condition and the f-trees satisfying it are called valid for Q.

Example 6. Consider the query $\pi_{A_T,C,D,E_T} Q_1$, where Q_1 is defined in Example 5. The attribute class $\{B_R, B_S\}$ is entirely projected out, so the relations R and S are now dependent, all their attributes are mutually dependent, and hence the corresponding nodes $\{A_T\}, \{C\}, \{D\}$ are dependent. The relation T induces the dependency of the nodes $\{A_T\}$ and $\{E_T\}$. The left f-tree in Figure 2 satisfies the path condition and hence is valid for our query, while the right f-tree, which is obtained by removing from the first f-tree in Figure 1 the attributes projected away, is not valid. $\square$

An alternative characterisation of the f-trees of a conjunctive query Q is via f-trees of its equi-join $\hat{Q}$. Given an f-tree $\hat{\mathcal{T}}$ of $\hat{Q}$, a first approach to compute an f-tree $\mathcal{T}$ of Q is to remove the attributes from $\hat{\mathcal{T}}$ that are projected away in Q. A problem arises when all attributes of a node are projected away: the node remains unlabelled, the expressions of the corresponding union would not be labelled by distinct singletons, and the resulting f-representation may encode duplicate products of singletons. Removing an empty node from the f-tree is also not always feasible as illustrated by Example 6: the attributes in its children subtrees may become dependent, which invalidates the f-tree. However, this problem does not arise when removing a leaf node. This observation leads to an alternative characterisation of f-trees of arbitrary conjunctive queries.

Definition 6. An *extension* of an f-tree $\mathcal{T}$ of a conjunctive query Q is an f-tree $\hat{\mathcal{T}}$ of the equi-join $\hat{Q}$ of Q such that $\mathcal{T}$ can be obtained from $\hat{\mathcal{T}}$ by erasing the non-head attributes in Q and repeatedly removing empty leaf nodes. $\square$

PROPOSITION 5. *An f-tree $\mathcal{T}$ is valid for a conjunctive query Q iff there exists an extension $\hat{\mathcal{T}}$ of $\mathcal{T}$.*

Example 7. The left f-tree in Figure 2 is valid for the query $\pi_{A_T,C,D,E_T} Q_1$ and can be extended to an f-tree $\hat{\mathcal{T}}$ for Q_1 by adding a leaf with attributes B_R, B_S under D, and adding the attributes A_R, A_S to the node labelled by A_T. The f-tree $\hat{\mathcal{T}}$ then satisfies the condition from Proposition 5.

The right f-tree in Figure 2 cannot be extended to an f-tree valid for Q_1, since the leaf B_R, B_S would have to be a descendant of C and also a descendant of D. $\square$

We finally note a connection between f-trees and path decompositions of the query hypergraph [19]. A path decomposition of a query Q is a path P whose nodes are bags of attribute classes of Q, such that

- for each attribute class of Q, the nodes of P containing that class form a non-empty connected path, and

- for each relation R in Q, there is a node in P that contains the classes of all attributes of R.

Proposition 6. *Each f-tree of an equi-join query corresponds to a path decomposition of its hypergraph.*

For queries with projection, this connection is lost since the query hypergraph cannot be recovered from the f-trees.

We next sketch the proof of Proposition 6. Consider a left-to-right order of the nodes in the f-tree $\mathcal{T}$ of the query Q. Let $L_1, \ldots, L_k$ be the leaves of $\mathcal{T}$ in this order and P_i be the set of attribute classes (i.e., nodes of $\mathcal{T}$) on the path from L_i to the root. For each attribute class $\mathcal{A}$, the set of indices i for which P_i contains $\mathcal{A}$ is a contiguous range of integers. Therefore, if we arrange the collections P_i in a path $P_1 - \cdots - P_k$, for each $\mathcal{A}$ the set of P_i containing $\mathcal{A}$ forms a connected path. Moreover, for any relation R in Q, the attributes of R lie on a root-to-leaf path in $\mathcal{T}$, so they are contained in P_i for some index i. Therefore, $P_1 - \cdots - P_k$ is a path decomposition P of the hypergraph of Q.

6. SIZE BOUNDS

The main result of this section is the following characterisation of conjunctive queries based on the size of f-representations of their results.

Theorem 1. *For any non-Boolean query $Q = \pi_{\mathcal{P}}\sigma_\psi(R_1 \times \cdots \times R_n)$ there is a rational number $s(Q)$ such that:*

- *For any database $\mathbf{D}$, there exists an f-representation of $Q(\mathbf{D})$ with size at most $|\mathcal{P}| \cdot |\mathbf{D}|^{s(Q)}$.*
- *For any f-tree $\mathcal{T}$ of Q, there exist arbitrarily large databases $\mathbf{D}$ for which the f-representation of $Q(\mathbf{D})$ over $\mathcal{T}$ has size at least $(|\mathbf{D}|/|Q|)^{s(Q)}$.*

The results of Boolean queries either consist of the nullary tuple or are the empty relation, both of size 1. In this section we only consider non-Boolean queries.

Under the assumption that the size $|Q|$ of the query Q and the size $|\mathcal{P}|$ of Q's projection list are constant, the bounds become tight and in $\Theta(|\mathbf{D}|^{s(Q)})$. The precise meaning of $s(Q)$ is given later in this section.

In the sequel, we first derive the size of the f-representation $\mathcal{T}(\mathbf{R})$ as a function of the relation $\mathbf{R}$ and the f-tree $\mathcal{T}$. We then derive upper and lower bounds on the size of the f-representation of a query result as functions of the sizes of the input database and query.

6.1 Counting Singletons in F-representations

Let $\mathbf{R}$ be a relation and $\mathcal{T}$ be an f-tree for which the f-representation $\mathcal{T}(\mathbf{R})$ exists. If C is an attribute in the root of $\mathcal{T}$, $\mathcal{T}(\mathbf{R})$ contains a single occurrence of the singleton $\langle C\!:\!c\rangle$ for each C-value c in $\mathbf{R}$. Now suppose B is an attribute in a child of the root, and $\mathcal{T}_B$ is the subtree of $\mathcal{T}$ rooted at B. For each C-value c, $\mathcal{T}(\mathbf{R})$ contains a subexpression over $\mathcal{T}_B$ which contains a singleton $\langle B\!:\!b\rangle$ for each B-value b in $\sigma_{C=c}\mathbf{R}$. Continuing top-down along $\mathcal{T}$, we deduce that for any attribute A, each singleton $\langle A\!:\!a\rangle$ appears once for each combination of values of the ancestor attributes of A, with which it contributes to some tuple of $\mathbf{R}$.

We next formalise the above observation and express the size of the f-representation $\mathcal{T}(\mathbf{R})$ as a function of $\mathbf{R}$ and $\mathcal{T}$.

Lemma 2. *Let $\mathcal{T}(\mathbf{R})$ be the f-representation of a relation $\mathbf{R}$ over an f-tree $\mathcal{T}$, A be an attribute of $\mathbf{R}$, and v be a value.*

- *The number of occurrences of the singleton $\langle A\!:\!v\rangle$ in $\mathcal{T}(\mathbf{R})$ is $|\pi_{\text{anc}(A)}\sigma_{A=v}\mathbf{R}|$.*
- *The number of occurrences of A-singletons in $\mathcal{T}(\mathbf{R})$ is $|\pi_{\text{path}(A)}\mathbf{R}|$.*
- $|\mathcal{T}(\mathbf{R})| = \sum_{A \textit{ attribute of } \mathbf{R}} |\pi_{\text{path}(A)}\mathbf{R}|$.

6.2 Upper Bounds

Lemma 2 gives an exact expression for the size of an f-representation $\mathcal{T}(\mathbf{R})$ in terms of the relation $\mathbf{R}$ and f-tree $\mathcal{T}$. In case $\mathbf{R}$ is the result $Q(\mathbf{D})$ of a non-Boolean query Q on a database $\mathbf{D}$, and $\mathcal{T}$ is an f-tree of Q, we can quantify the size of the f-representation $\mathcal{T}(Q(\mathbf{D}))$ directly in terms of the database size $|\mathbf{D}|$ as follows.

Recall from Lemma 2 that for any attribute A in $\mathcal{T}$, the number of singletons of type A is $|\pi_{\text{path}(A)}Q(\mathbf{D})|$. Define the equi-join query Q_A to be Q restricted to the attributes in path(A). That is, $Q_A = \sigma_{\psi_A}(R_1^A \times \cdots \times R_n^A)$, where ψ_A and R_i^A are ψ and R_i respectively, restricted to the attributes in path(A). Define also $\mathbf{D}_A$ to be the database $\mathbf{D}$ projected onto the attributes in path(A).

Lemma 3. *For any database $\mathbf{D}$, the number of occurrences of A-singletons in the f-representation $\mathcal{T}(Q(\mathbf{D}))$ is at most $|Q_A(\mathbf{D}_A)|$.*

We can further estimate the size of $Q_A(\mathbf{D}_A)$ as a function of $|\mathbf{D}_A|$ and Q_A. Intuitively, if we can cover all attributes of the query Q_A by $k \le |Q_A|$ of their relations, then $|Q_A(\mathbf{D}_A)|$ is at most the product of the sizes of these relations, which is at most $|\mathbf{D}_A|^k$. This corresponds to an edge cover of size k in the hypergraph of Q_A. The following result strengthens this idea by lifting covers to a weighted version [1].

Definition 7. For a query $Q = \sigma_\psi(R_1 \times \cdots \times R_n)$, the *fractional edge cover number* $\rho^*(Q)$ is the cost of an optimal solution to the linear program with variables $\{x_{R_i}\}_{i=1}^n$:

$$\begin{array}{lll} \text{minimise} & \sum_i x_{R_i} & \\ \text{subject to} & \sum_{i:R_i \in \text{rel}(\mathcal{A})} x_{R_i} \ge 1 & \text{for each attribute class } \mathcal{A}, \\ & x_{R_i} \ge 0 & \text{for all } i. \end{array}$$

For each relation R_i (or edge in the query hypergraph), its weight is given by the variable x_{R_i}. Each attribute class $\mathcal{A}$ (or each vertex in the query hypergraph) has to be covered by relations with attributes in $\mathcal{A}$ such that the sum of the weights of these relations is greater than 1. The objective is to minimise the sum of the weights of all relations. In the non-weighted version of the edge cover, the variables x_{R_i} can only be assigned the values 0 and 1, whereas in the weighted version the variables can hold any positive rational number. The following lemma improves the upper bound of $|\mathbf{D}_A|^k$ on $|Q_A(\mathbf{D}_A)|$ by shifting from a (minimal) edge covering number k to its fractional version.

Lemma 4 ([1]). *For any equi-join query Q and database $\mathbf{D}$, we have $|Q(\mathbf{D})| \le |\mathbf{D}|^{\rho^*(Q)}$.*

Together with Lemma 3, this yields the following bound.

Corollary 1. *For any database $\mathbf{D}$, the number of occurrences of A-singletons in the f-representation $\mathcal{T}(Q(\mathbf{D}))$ is at most $|\mathbf{D}|^{\rho^*(Q_A)}$.*

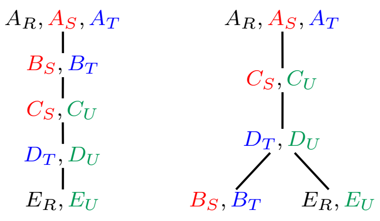

Figure 3: F-trees $\mathcal{T}_3$ and $\mathcal{T}_4$ of query Q_2 in Example 8.

Corollary 1 gives an upper bound on the number of occurrences of singletons of any attribute. By summing these bounds over all attributes of Q, we obtain an upper bound on the total number of occurrences of singletons in the f-representation $\mathcal{T}(Q(\mathbf{D}))$, that is, on the size of $\mathcal{T}(Q(\mathbf{D}))$. Define $s(\mathcal{T}) = \max_{A\in\mathcal{P}} \rho^*(Q_A)$ to be the maximal possible $\rho^*(Q_A)$ over all head attributes $A \in \mathcal{P}$ from Q. The bound on the size of $\mathcal{T}(Q(\mathbf{D}))$ becomes as follows:

Corollary 2. *The size of the f-representation $\mathcal{T}(Q(\mathbf{D}))$ is at most $|\mathcal{P}| \cdot |\mathbf{D}|^{s(\mathcal{T})}$.*

Let us now define $s(Q) = \min_{\mathcal{T}} s(\mathcal{T})$ to be the minimum possible $s(\mathcal{T})$ over all f-trees $\mathcal{T}$ valid for Q. We then have:

Corollary 3. *For any database $\mathbf{D}$, there exists an f-representation of $Q(\mathbf{D})$ with size at most $|\mathcal{P}| \cdot |\mathbf{D}|^{s(Q)}$.*

Example 8. Consider a database with relations R, S, T, and U with schemas $\{A_R, E_R\}$, $\{A_S, B_S, C_S\}$, $\{A_T, B_T, D_T\}$ and $\{C_U, D_U, E_U\}$ respectively, and the query $Q_2 = \sigma_\psi(R \times S \times T \times U)$, with $\psi = (A_R = A_S = A_T, B_S = B_T, C_S = C_U, D_T = D_U, E_R = E_U)$.

We compute $|\mathcal{T}_3(Q_2(\mathbf{D}))|$, where $\mathcal{T}_3$ is the left f-tree in Figure 3. For any attribute $E \in \{E_R, E_U\}$, path(E) contains all attributes of Q_2 and hence $Q_2^E = Q_2$. The attribute classes of this query can be covered by the two relations S and U, so $\rho^*(Q_2^E) \leq 2$. On the other hand, the attribute classes $\{B_S, B_T\}$ and $\{E_R, E_U\}$ have no relations in common, so their corresponding conditions $x_S + x_T \geq 1$ and $x_R + x_U \geq 1$ imply $\rho^*(Q_2^E) \geq 2$. We thus obtain $\rho^*(Q_2^E) = 2$. For each of the other attributes, their corresponding path is a subset of path(E), so their fractional cover numbers are at most 2. It follows that $s(\mathcal{T}_3) = 2$ and $|\mathcal{T}_3(Q_2(\mathbf{D}))| \leq 11 \cdot |\mathbf{D}|^2$ for any database $\mathbf{D}$. However, the same bound already holds for $|Q_2(\mathbf{D})|$, so factorising over $\mathcal{T}_3$ is not effective.

We now compute $|\mathcal{T}_4(Q_2(\mathbf{D}))|$, where $\mathcal{T}_4$ is the right f-tree in Figure 3. The nodes with largest paths are $\{B_S, B_T\}$ and $\{E_R, E_U\}$. Let us pick any attribute B and respectively E from the two attribute classes and consider the query restrictions Q_2^B and Q_2^E. We need at least two relations to cover all attributes of Q_2^B, so the edge cover number of Q_2^B is 2. However, in the fractional edge cover linear program, we can assign $x_S = x_T = x_U = 1/2$ and $x_R = 0$. The covering conditions are satisfied, since each attribute class is covered by two of the relations S, T, U. The cost of this solution is 3/2. It is in fact the optimal solution, so $\rho^*(Q_2^B) = 3/2$. For Q_2^E, the optimal solution is $x_U = 2/3$ and $x_R = x_S = x_T = 1/3$ with total cost $\rho^*(Q_2^E) = 5/3$, and hence $s(\mathcal{T}_4) = 5/3$. Thus by using $\mathcal{T}_4$, which unlike $\mathcal{T}_3$ separates $\{B_S, B_T\}$ and $\{E_R, E_U\}$ into independent branches, we obtain factorisations $\mathcal{T}_4(Q_2(\mathbf{D}))$ with size at most $11 \cdot |\mathbf{D}|^{5/3} \ll 11 \cdot |\mathbf{D}|^2$. In fact, $\mathcal{T}_4$ is an optimal f-tree and $s(Q_2) = 5/3$. □

6.3 Lower Bounds

We next show that the upper bound on the f-representation size is tight. For any non-Boolean query Q and any f-tree $\mathcal{T}$ of Q, there are arbitrarily large databases for which the size of the f-representation of the query result over $\mathcal{T}$ asymptotically meets the upper bound.

Following Lemma 3, the number of occurrences of A-singletons is at most $|Q_A(\mathbf{D}_A)|$. In a first attempt to construct a database $\mathbf{D}_A$ with large result for the query Q_A, we pick k attribute classes of Q_A and let each of them attain N different values. If each relation has attributes from at most one of these classes and size at most N, then the result of the query Q_A has size N^k. These k attribute classes correspond to an independent set of k nodes in the hypergraph of Q_A.

Similar to the upper bound, we can strengthen the above lower bound by lifting independent sets to a weighted version. Since the linear programs for the (fractional) edge cover and the independent set problems are dual, this lower bound meets the upper bound from Section 6.2. The following result forms the basis of our argument (our proof extends the one in [1] to queries with repeating relation symbols).

Lemma 5. *For any equi-join query Q, there exist arbitrarily large databases $\mathbf{D}$ such that $|Q(\mathbf{D})| \geq (|\mathbf{D}|/|Q|)^{\rho^*(Q)}$.*

We now use Lemmata 2 and 5 to derive lower bounds on the number of occurrences of singletons of an attribute A in the f-representation of $Q(\mathbf{D})$ over any f-tree $\mathcal{T}$ of Q.

Lemma 6. *There exist arbitrarily large databases $\mathbf{D}$ such that the number of occurrences of A-singletons in the f-representation $\mathcal{T}(Q(\mathbf{D}))$ is at least $(|\mathbf{D}|/|Q|)^{\rho^*(Q_A)}$.*

We now lift Lemma 6 from A-singletons to all singletons in $\mathcal{T}(Q(\mathbf{D}))$ by considering the attribute A for which the lower bound $(|\mathbf{D}|/|Q|)^{\rho^*(Q_A)}$ is the largest.

Corollary 4. *There exist arbitrarily large databases $\mathbf{D}$ for which the size of the f-representation $\mathcal{T}(Q(\mathbf{D}))$ is at least $(|\mathbf{D}|/|Q|)^{s(\mathcal{T})}$.*

Finally, by minimising over all f-trees $\mathcal{T}$ valid for Q, we find a lower bound on the size of the f-representation of $Q(\mathbf{D})$ over f-trees.

Corollary 5. *For any f-tree $\mathcal{T}$ of Q there exist arbitrarily large databases $\mathbf{D}$ for which the f-representation $\mathcal{T}(Q(\mathbf{D}))$ has size at least $(|\mathbf{D}|/|Q|)^{s(Q)}$.*

For a fixed query and schema, the upper and lower bounds on the size of f-representations of query results meet asymptotically due to duality of linear programming. The fractional versions of the minimum hyperedge cover number for the upper bounds and of the maximum independent set number for the lower bounds are essential for the tightness result, since their integer versions need not be equal.

The parameter $s(Q)$ thus completely characterises queries by the factorisability of their results within the class of f-representations defined by f-trees.

Example 9. Let us continue Example 8 and consider the query Q_2 and the right f-tree $\mathcal{T}_4$ from Figure 3. The hypergraph of Q_2^E, where $E \in \{E_R, E_U\}$, has maximum independent set of size 1, since any two nodes share a common edge.

We can trivially construct databases $\mathbf{D}$ for which the number of E-singletons is linear in $\mathbf{D}$, yet this is much smaller than the lower bound given by Corollary 5. The fractional relaxation of the maximum independent set problem allows to increase the optimal cost to 5/3, thus meeting $\rho^*(Q_2^E)$ by duality of linear programming. In this relaxation we assign nonnegative values to the attribute classes, so that the sum of values in each relation is at most one. By assigning $y_{\mathcal{A}} = 2/3$ and $y_{\mathcal{C}} = y_{\mathcal{D}} = y_{\mathcal{E}} = 1/3$, the sum in each relation is exactly one, and the total cost is 5/3. This is used in the proofs of Lemmas 5 and 6 to construct arbitrarily large databases $\mathbf{D}$ for which the number of E-singletons in $\mathcal{T}_4(Q_2(\mathbf{D}))$ is at least $(|\mathbf{D}|/|Q_2|)^{5/3} = (|\mathbf{D}|/4)^{5/3}$.

One such database $\mathbf{D}$ would contain the relations $\mathbf{R} = [4]\times[2]$, $\mathbf{S} = [4]\times[2]\times[1]$, $\mathbf{T} = [4]\times[2]\times[1]$ and $\mathbf{U} = [2]\times[2]\times[2]$. Here $[N]$ denotes $\{1,\ldots,N\}$ and the attributes of each relation are ordered alphabetically. Each relation has size 8 and the database $\mathbf{D}$ has size $32 = 8\times|Q_2|$. The result $Q(\mathbf{D})$ corresponds to the relation where $A_R = A_S = A_T \in [4]$, $B_S = B_T = 1$, $C_S = C_U \in [2]$, $D_T = D_U \in [2]$ and $E_R = E_U \in [2]$, and any combination of these values is allowed. Its size is $|Q_2(\mathbf{D})| = 32 = (32/4)^{5/3} = (|\mathbf{D}|/|Q_2|)^{5/3}$.

Since all f-trees $\mathcal{T}$ for Q_2 have $s(\mathcal{T}) \geq s(Q_2) = 5/3$, the results in this subsection show that for any such f-tree $\mathcal{T}$ we can find databases $\mathbf{D}$ for which the size of $\mathcal{T}(Q_2(\mathbf{D}))$ is at least $(|\mathbf{D}|/|Q_2|)^{5/3} = (|\mathbf{D}|/4)^{5/3}$. □

6.4 Succinctness Gap

Corollary 3 shows that a shift from standard tabular representations of query results to equivalent factorised representations can bring an exponential gain in representation size. For equi-join queries, this gain is precisely captured by the difference between the exponent $\rho^*(Q)$, which defines tight bounds on the size of tabular representations of query results [1], and the exponent $s(Q) = \min_{\mathcal{T}}(\max_A(\rho^*(Q_A)))$, which defines tight bounds on the size of factorised representations of query results. We note that $\rho^*(Q)$ has been previously defined for equi-join queries only [1], whereas $s(Q)$ is defined for arbitrary conjunctive queries. By definition of $s(Q)$, we have that $s(Q) \leq \rho^*(Q)$ for any query Q.

The gap between $\rho^*(Q)$ and $s(Q)$ can be as much as $|Q|-1$ and thus arbitrarily large: if Q is a product of relations, then $\rho^*(Q) = |Q|$ and $s(Q) = 1$. Any query, where at least one attribute per relation is not involved in joins, has $\rho^*(Q) = |Q|$ yet it may still retain $s(Q) = 1$. Equi-join queries Q whose Boolean projections $\pi_{\emptyset}Q$ are hierarchical, and queries whose head attributes belong to one equivalence class have $s(Q) = 1$. An arbitrarily large gap can be also witnessed by shifting from edge covers to fractional edge covers to characterise bounds on the cardinality of results for equi-join queries [10].

We next exemplify a class of queries with $s(Q) > 1$, but for which the succinctness gap is still exponential.

Example 10. Consider relations R_i over schemas $\{A_i, B_i\}$ $\forall 1 \leq i \leq n$. Let $\psi = \bigwedge_{i=1}^{n-1}(B_i = A_{i+1})$ and $Q_n = \sigma_\psi(R_1 \times \cdots \times R_n)$. This query is a chain of $n-1$ joins.

An f-tree $\mathcal{T}$ for Q_{11} (i.e., $n = 11$) is shown in Figure 4. Let Q_{A_1} be the restriction of Q_{11} to path(A_1). Then, $\rho^*(Q_{A_1}) = 3$. This value is 3 or less for other attributes, so $s(\mathcal{T}) = 3$, which is the lowest possible value and hence $s(Q_{11}) = 3$. For arbitrary n, $s(Q_n) = \Theta(\log n)$. The fractional edge cover of Q_n has cost $\Theta(n)$, so by Lemma 5, the query result $Q_n(\mathbf{D})$ can be as large as $\Omega((|\mathbf{D}|/n)^{\Theta(n)})$. This is exponentially larger than the upper bound $2n\cdot|\mathbf{D}|^{\Theta(\log n)}$ on the size of its equivalent f-representation over an f-tree witnessing $s(Q_n)$.□

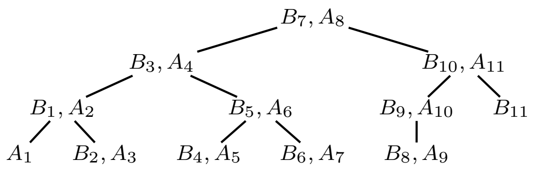

Figure 4: An f-tree for Q_{11} with $s(\mathcal{T}) = 3$, the lowest possible over all f-trees of Q_{11} (From Example 10).

Example 10 shows that branching in f-trees is key to a low value for the parameter $s(Q)$ and thus to succinct f-representations. Equi-join queries Q whose Boolean projections $\pi_{\emptyset}Q$ are hierarchical admit f-trees with maximal branching factor and hence value 1 for $s(Q)$: for each root-to-leaf path in such an f-tree there is a relation with attributes in each node of the path. If branching is not possible and the f-tree is a single path, then the factorisation over such an f-tree is in worst case as large as its equivalent tabular representation. Example 8 shows this for the left f-tree $\mathcal{T}_3$ in Figure 3. This observation suggests that queries whose f-trees are paths cannot benefit from factorisations, since their results are in general not factorisable. All f-trees of a query Q are paths if and only if any two nodes are dependent. For equi-join queries this means that any two attribute classes have attributes from a common relation, as exemplified next.

Example 11. Consider the relations $R_{i,j}$ for $1 \leq i < j \leq n$ with schemas $\{A_{i,j}^i, A_{i,j}^j\}$. Let $Q = \sigma_\psi(\times_{i<j} R_{i,j})$, where ψ equates all attributes with the same superscript i.

The f-trees of the query Q have n nodes that correspond to attribute classes in Q, and for each pair of nodes a relation containing one attribute from both nodes. Therefore, the possible $(n!)$ f-trees of Q are paths of n nodes.

For each such tree $\mathcal{T}$, the path of an attribute A in the bottom node includes all nodes of $\mathcal{T}$, the associated query Q_A is equal to Q, and its fractional edge cover number is $\rho^*(Q) = \binom{n}{2}\frac{1}{n-1} = \frac{n}{2}$. (An optimal fractional edge cover assigns weight $\frac{1}{n-1}$ to each of the $\binom{n}{2}$ relations.) It follows that $s(\mathcal{T}) = \frac{n}{2}$ for any f-tree $\mathcal{T}$ of Q, and hence $s(Q) = \frac{n}{2}$.□

7. FINDING THE EXPONENT

In this section we give an algorithm to compute the parameter $s(Q)$ that characterises the upper bound on the size of f-representations of query results. The algorithm iterates with polynomial delay over some f-trees of the input query Q, including an f-tree $\mathcal{T}$ with minimum $s(\mathcal{T})$. For each f-tree $\mathcal{T}$ we then compute in polynomial time the parameter $s(\mathcal{T})$, which is the maximum $\rho^*(Q_A)$ over all attributes A in Q; in contrast, finding the integer version of $\rho^*(Q_A)$ is NP-hard. The parameter $s(Q)$ is the minimum $s(\mathcal{T})$ over all enumerated f-trees $\mathcal{T}$.

An f-tree $\mathcal{T}$ valid for a query Q is an unordered rooted forest whose nodes are labelled by the equivalence classes of head attributes of Q and satisfy the path condition. In case $\mathcal{T}$ is a tree with root $\mathcal{A}$ and subtrees $\mathcal{U}$, then $\mathcal{T}$ satisfies the path condition iff $\mathcal{U}$ satisfies the path condition. In case $\mathcal{T}$ is a forest $\mathcal{T}_1,\ldots,\mathcal{T}_n$, then it satisfies the path condition iff

```
Let $D_\mathcal{A}$ be the set of nodes dependent on node $\mathcal{A}$
Let $>$ be a partial order on nodes: $\mathcal{A} > \mathcal{B}$ if
  • $D_\mathcal{B} \subset D_\mathcal{A}$, or
  • $D_\mathcal{B} = D_\mathcal{A}$ and $\mathcal{A}$ comes before $\mathcal{B}$ in lexicographic order.

iter (set $S$ of nodes)
  if $S$ is empty then
    output the empty tree
  else let $P_1, \ldots, P_n$ be a maximal partition of $S$ such that
    for all $i \neq j$, any $\mathcal{A} \in P_i$ and $\mathcal{B} \in P_j$ are independent
    if $n = 1$ then
      foreach $>$-maximal $\mathcal{A} \in S$ do                    (*)
        foreach $\mathcal{U} \in$ iter($S \setminus \{\mathcal{A}\}$) do
          output $\mathcal{A}(\mathcal{U})$
    else
      foreach $(\mathcal{T}_1, \ldots, \mathcal{T}_n) \in$ (iter($P_1$), ..., iter($P_n$)) do
        output $\mathcal{T}_1 \cup \cdots \cup \mathcal{T}_n$
```

Figure 5: Enumerating f-trees of a query. The set S consists of the nodes labelled by the equivalence classes of the head attributes in the query.

each of the trees $\mathcal{T}_1, \ldots, \mathcal{T}_n$ satisfies the path condition and the nodes in different trees are independent.

This characterisation of the path condition suggests a recursive algorithm for enumerating all valid f-trees over a set S of nodes: try all possible partitions of S into independent trees, for each tree try all possible nodes as root and continue recursively for its descendants.

Example 12. Consider the query Q_1 in Example 5. Any valid f-tree must be a single tree since the node $\{A_R, A_S, A_T\}$ is dependent on all other nodes. If we choose $\{A_R, A_S, A_T\}$ for a root, in the next recursive call we can split the remaining nodes into $P_1 = \{\{B_R, B_S\}, \{C\}, \{D\}\}$ and $P_2 = \{E_T\}$. The nodes in P_1 and P_2 are independent since R and S only have attributes in P_1 and T only has attributes in P_2. The f-tree $\mathcal{T}_1$ in Figure 1 is produced within this recursive call. If we choose $\{B_R, B_S\}$ for a root, in the next recursive call the remaining nodes cannot be split into independent subtrees, since the $\{A_R, A_S, A_T\}$ is dependent on all other nodes. In Figure 1, the f-tree $\mathcal{T}_2$ is produced within this call, while the third tree is not a valid f-tree and thus not produced. □

Some choices of partitions and of roots are always suboptimal when searching for an f-tree $\mathcal{T}$ with lowest possible $s(\mathcal{T})$. We next discuss two improvements that are incorporated in the algorithm **iter** given in Figure 5.

Firstly, it always pays off to have as root a node which is dependent on as many other nodes as possible.

Lemma 7. *Let $\mathcal{T}$ be an f-tree with nodes $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{B}$ is an ancestor of $\mathcal{A}$ and all nodes dependent on $\mathcal{B}$ are also dependent on $\mathcal{A}$. By exchanging $\mathcal{A}$ and $\mathcal{B}$ in $\mathcal{T}$ we do not violate the path condition and do not increase $s(\mathcal{T})$.*

Lemma 7 implies that we do not need to consider f-trees with root $\mathcal{B}$. For example, the f-tree $\mathcal{T}_2$ in Figure 1 is suboptimal since $\{B_R, B_S\}$ is the root instead of $\{A_R, A_S, A_T\}$. In addition, if the nodes $\mathcal{A}$ and $\mathcal{B}$ are dependent on the same nodes, they are interchangeable in any f-tree (with respect to the path condition), and to find an optimal f-tree we only need to consider one of them as root.

Secondly, it always pays off to partition the nodes in as many independent trees as possible. For any set of nodes, there always exists a maximal partition such that the nodes are independent across parts. For any coarser partition, we could further partition while not increasing path(A) for any attribute A and thus not increasing $s(\mathcal{T})$.

Example 13. For the query Q_1 in Example 5, the algorithm **iter** in Figure 5 does not output the second tree in Figure 1. The node $\{B_R, B_S\}$ is not considered for the root since it depends on a strict subset of the nodes that $\{A_R, A_S, A_T\}$ depends on. In fact, for Q_2 **iter** only produces the first tree from Figure 1. □

Using lazy evaluation, at any one time the number of calls of **iter** on the stack is at most linear in the number of nodes, and between two consecutively generated f-trees, there are at most linearly many recursive calls. The execution time inside each call is dominated by the computation of the maximal partition (takes quadratic time in the number of nodes in S if node dependencies are precomputed), and the iteration over all $>$-maximal nodes (takes quadratic time in the number of nodes in S if the $>$-order is precomputed). The delay between consecutive f-trees is thus at most cubic in the number of nodes. The precomputation of node dependencies takes time quadratic in the number of attributes and the precomputation of the $>$-order takes time cubic in the number of nodes. The space needed for both operations is quadratic in the number of nodes.

Theorem 2. *Given a query Q, Algorithm* **iter**
- *enumerates f-trees of Q with delay cubic and space quadratic in the number of Q's equivalence classes and*
- *finds an f-tree $\mathcal{T}$ with $s(\mathcal{T}) = s(Q)$.*

The algorithm can enumerate exponentially many f-trees.

Example 14. The query Q in Example 11 has n attribute classes and thus n nodes in any f-tree of Q. These nodes cannot be partitioned by **iter** since any two of them contain an attribute of a relation $R_{i,j}$ and hence all are dependent. Therefore the valid f-trees for Q are paths of the n nodes. The algorithm **iter** outputs $n!$ distinct f-trees of Q. □

For a given f-tree $\mathcal{T}$, we can compute $s(\mathcal{T})$ in polynomial time. In this sense we can recognise good f-trees efficiently. In particular, given a query Q and a positive rational number q, the threshold problem of determining whether Q has an f-tree $\mathcal{T}$ with $s(\mathcal{T}) \leq q$ is in NP: the certificate is $\mathcal{T}$ itself.

The parameter $s(Q)$ is the lowest value q for which there exists an f-tree $\mathcal{T}$ with $s(\mathcal{T}) \leq q$. Given an NP oracle answering the threshold problem above, we can find $s(Q)$ by binary search over the rationals (using the Stern-Brocot tree). Since the value of $s(Q)$ is of polynomial length, the search makes only polynomially many queries to the oracle. It remains open whether the threshold problem is NP-hard.

8. COMPUTING F-REPRESENTATIONS OF QUERY RESULTS

The algorithm **gen** in Figure 6 computes the f-representation of a query result over a given f-tree of the query. A key feature of this algorithm is that it does not require the query

```
gen(f-tree $\mathcal{T}$, ranges of tuples $(\mathcal{R}_1, \dots, \mathcal{R}_n)$)
 if $\exists i : \mathcal{R}_i = \emptyset$ then return $\emptyset$
 if $\mathcal{T}$ is empty  then return $\langle\rangle$
 if $\mathcal{T}$ is a forest $\mathcal{T}_1, \dots, \mathcal{T}_k$ then
     return gen($\mathcal{T}_1, (\mathcal{R}_1, \dots, \mathcal{R}_n)$) $\times \dots \times$
            gen($\mathcal{T}_k, (\mathcal{R}_1, \dots, \mathcal{R}_n)$)
 if $\mathcal{T}$ is a tree $\mathcal{A}(\mathcal{U})$ then
     foreach $1 \le j \le n$ do let $\mathcal{R}'_j = \mathcal{R}_j$
     let $E = \emptyset$
     foreach value $a$ shared by all attributes in $\mathcal{A}$ do     (*)
         let $F = \langle\rangle$
         foreach attribute $A \in \mathcal{A}$ do
             let $\mathcal{R}_i$ = range of relation with attribute $A$
             let $\mathcal{R}'_i$ = the subrange of $\mathcal{R}_i$ with $A = a$
             if $A$ is a head attribute then $F = F \times \langle A{:}a\rangle$
                                      else $F = F \times \langle\rangle$
         let $E = E \cup F \times$ gen($\mathcal{U}, (\mathcal{R}'_1, \dots, \mathcal{R}'_n)$)
     return $E$
```

Figure 6: Computing the f-representation of the result of a query over a given f-tree.

result as input and constructs the f-representation directly from the input database. This is desirable since the query result can be much larger than its f-representation.

Given a query Q, an f-tree $\mathcal{T}_0$ of Q, and a database D, the algorithm takes as input (1) an f-tree $\mathcal{T}$ that is an extension of $\mathcal{T}_0$ and thus captures all joins in Q, and (2) a list of ranges of tuples in each relation in the database $\mathbf{D}$; initially, each range $\mathcal{R}_i$ for a relation $\mathbf{R}_i$ is $(1, |\mathbf{R}_i|)$ and can thus address all tuples in $\mathcal{R}_i$. We assume that the attributes of each relation $\mathbf{R}_i$ are previously ordered following a topological sort of $\mathcal{T}$ and each relation $\mathbf{R}_i$ is sorted lexicographically by the order of its attributes.

The nodes of the f-tree $\mathcal{T}$ are processed top-down, using recursion on the structure of $\mathcal{T}$. If we sort the tuples of each relation by attributes according to a topological ordering of $\mathcal{T}$'s nodes, then in each call to **gen** the range of tuples forms a contiguous range $\mathcal{R}_i$ in each relation $\mathbf{R}_i$ and we only need to pass pointers to the beginning and end of $\mathcal{R}_i$.

If an input range is empty, then it represents the empty relation and thus the whole result is empty. If no range is empty and the f-tree $\mathcal{T}$ is empty, then the represented relation is not empty and must be the nullary singleton $\langle\rangle$. In case $\mathcal{T}$ is a forest, we then recurse on each of its trees with the same input database given as a list of tuple ranges.

At each call with a tree rooted at a node $\mathcal{A}$, for each relation $\mathbf{R}_i$ with an attribute $A \in \mathcal{A}$, the tuples of $\mathbf{R}_i$ in the corresponding range $\mathcal{R}_i$ are already sorted by A. When iterating over values a common to all attributes in $\mathcal{A}$, the tuples with value a for each attribute in $\mathcal{A}$ appear in sequence and the values common to all attributes can be found by one scan over all ranges. This defines the new tuple ranges $\mathcal{R}'_i$ and we recurse on the children of the current node with the new ranges. The ranges of relations without attributes in $\mathcal{A}$ remain unchanged. For each value a, we construct a product of the f-representation returned by the recursive call on the children of the current node and of the singletons for the attributes in $\mathcal{A}$. In case of queries with projection, there may be attributes that are not head. We accommodate this by considering nullary singletons instead of singletons for non-head attributes. This corresponds to projecting out the singletons corresponding to the non-head attributes.

The obtained f-representation is not necessarily normalised, since during its construction we may introduce nullary singletons and empty relations. The f-representation can be normalised in linear time as discussed in Section 4.

Proposition 7. *Let Q be a query, $\mathcal{T}$ be an f-tree of Q, $\hat{\mathcal{T}}$ be an extension of $\mathcal{T}$, and $\mathbf{D} = (\mathbf{R}_1, \dots, \mathbf{R}_n)$ be a database. The f-representation $\mathcal{T}(Q(\mathbf{D}))$ can be computed by* **gen**$(\hat{\mathcal{T}}$, $((1, |\mathbf{R}_1|), \dots, (1, |\mathbf{R}_n|)))$ *followed by normalisation.*

In case of a query with projection, during the top-down processing of the nodes of the f-tree $\mathcal{T}$ we can reach a node without head attributes. To compute the f-representation of the query result, we only need to check at that node for the emptiness of the query associated with that node on the database defined by the ranges valid at that recursion step. This can be done using any existing query evaluation method for Boolean queries [11]. The (suboptimal) approach considered here is to compute the f-representation Φ of the result of that query, normalise it, and then check for emptiness. An optimisation is to avoid computing the whole f-representation Φ and stop once the represented relation contains a (nullary) tuple.

We next investigate the time complexity of the algorithm first for equi-joins and then for arbitrary conjunctive queries.

The time taken by calls in case $\mathcal{T}$ is a forest is absorbed into subsequent calls for trees. Consider now a call in case $\mathcal{T}$ is a tree rooted at a node $\mathcal{A}$. During the execution of the loop over values a, for each relation $R_i \in \text{rel}(\mathcal{A})$ the tuples in the range $\mathcal{R}_i$ have equal values for attributes in anc$(\mathcal{A})$. The next attribute of each R_i in the topological sort of $\mathcal{T}$ is an attribute from $\mathcal{A}$, so each range $\mathcal{R}_i$ is sorted by the values of the attribute from $\mathcal{A}$. Therefore, to iterate over all ranges $\mathcal{R}'_i \subseteq \mathcal{R}_i$ sharing the same value of $\mathcal{A}$, it suffices to scan these ranges simultaneously, searching for common values of $\mathcal{A}$. The time complexity of this iteration is linear in the total number of tuples in the ranges $\mathcal{R}_i$ for relations $R_i \in \text{rel}(\mathcal{A})$. The following lemma quantifies this number.

Lemma 8. *Given an f-tree $\mathcal{T}$ of an equi-join query Q, the number of tuples visited by* **gen** *when computing $\mathcal{T}(Q(\mathbf{D}))$ is $O(|\mathcal{S}| \cdot |Q| \cdot |\mathbf{D}|^{s(\mathcal{T})})$.*

An additional cost is sorting the relations of $\mathbf{D}$ before calling **gen**. This takes time $O(|\mathcal{S}| \cdot |\mathbf{D}| \cdot \log |\mathbf{D}|)$. Together with Lemma 8, this yields an upper bound on the running time of our factorisation algorithm.

Theorem 3. *Let Q be an equi-join query. For any f-tree $\mathcal{T}$ of Q and database $\mathbf{D}$, the f-representation $\mathcal{T}(Q(\mathbf{D}))$ can be computed in time*

- *$O(|\mathcal{S}| \cdot |\mathbf{D}| \cdot (|Q| + \log |\mathbf{D}|))$ if $s(\mathcal{T}) = 1$, and*
- *$O(|\mathcal{S}| \cdot |Q| \cdot |\mathbf{D}|^{s(\mathcal{T})})$ otherwise.*

For equi-join queries, the time to compute the f-representation is thus near-optimal, since it only needs an additional factor of schema size ($|\mathcal{S}|$) to the size of the f-representation.

This result carries over to arbitrary conjunctive queries by considering extensions $\hat{\mathcal{T}}$ of $\mathcal{T}$. That is, we construct the f-representation of the query result by calling our algorithm

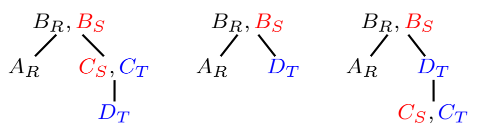

Figure 7: Left to right: an f-tree $\mathcal{T}_5$ for the query Q_3, an f-tree $\mathcal{T}_6$ for the query $\pi_{A_R,B_R,B_S,D_T}Q_3$ in Example 15, and its optimal extension $\hat{\mathcal{T}}_6$.

R	B_R	A_R	S	B_S	C_S	T	C_T	D_T
✓	**2**	**1**	✓	**2**	**1**	✓	**1**	**2**
	3	1		3	1	✓	**1**	**3**
	3	2		3	2	✓	**2**	**3**

Figure 8: A database $\mathbf{D}$ during the execution of gen$(\mathcal{T}_5, Q_3, \mathcal{R})$.

with parameter $\hat{\mathcal{T}}$ and projecting out singletons corresponding to non-head attributes. To improve performance, we consider *optimal* extensions.

Definition 8. An extension $\hat{\mathcal{T}}$ of an f-tree $\mathcal{T}$ is *optimal* if $s(\hat{\mathcal{T}})$ is minimal among all extensions of $\mathcal{T}$. □

The time complexity of computing f-representations in case of conjunctive queries depends on optimal extensions of their f-trees. This can incur a considerable overhead, since their optimal extensions can be arbitrarily larger. For instance, Boolean queries have empty f-trees with optimal extensions of size linear in the number of equivalence classes. This overhead accounts however for the hardness of checking emptiness for conjunctive queries. The following theorem strictly generalises Theorem 3.

THEOREM 4. *Let Q be a conjunctive query, $\mathcal{T}$ be an f-tree of Q, and $\hat{\mathcal{T}}$ be an optimal extension of $\mathcal{T}$. For any database* $\mathbf{D}$, *Algorithm* **gen** *computes the f-representation $\mathcal{T}(Q(\mathbf{D}))$ in time*

- $O(|\mathcal{S}| \cdot |\mathbf{D}| \cdot (|Q| + \log |\mathbf{D}|))$ *if $s(\hat{\mathcal{T}}) = 1$, and*
- $O(|\mathcal{S}| \cdot |Q| \cdot |\mathbf{D}|^{s(\hat{\mathcal{T}})})$ *otherwise.*

Example 15. Consider the relations R, S and T over schemas $\{A_R, B_R\}$, $\{B_S, C_S\}$ and $\{C_T, D_T\}$, and the query $Q_3 = \sigma_\psi(R \times S \times T)$ with $\psi = (B_R = B_S \wedge C_S = C_T)$. Figure 7 depicts an f-tree $\mathcal{T}_5$ of Q_3. Let $\mathcal{B}$ be the node with the equivalence class $\{B_R, B_S\}$, and similarly for $\mathcal{A}$, $\mathcal{C}$ and $\mathcal{D}$.

Let us examine the execution of the call **gen**$(\mathcal{T}_5, \mathcal{R})$, where $\mathcal{R}$ represents the full range in each relation of database $\mathbf{D}$ in Figure 8. The attributes of each relation are already ordered following a topological sort of $\mathcal{T}_5$ in Figure 8. The root of $\mathcal{T}_5$ is the node $\mathcal{B}$, and the first execution of the loop (*) finds the ranges marked by (✓) in Figure 8, with the common value of $B_R = B_S = 2$. Notice that T does not have an attribute in the root node, so its range remains unchanged. After these ranges are found, they are passed to a next call of **gen** on the subtree formed by the children of $\mathcal{B}$, which recurses separately into the subtree formed by $\mathcal{A}$ and the subtree formed by $\mathcal{C}$ and $\mathcal{D}$. The latter call iterates over the common values C_S and C_T of S and T respectively within the current ranges. The only common value found is 1, for which the range of T is restricted to the first two tuples and **gen** is called on the leaf $\mathcal{D}$. The second execution of the loop (*) in the outermost call of **gen** finds ranges with common value $B_R = B_S = 3$ and again recurses into the subtrees below $\mathcal{B}$. The final result of the algorithm is

$$\begin{aligned}\mathcal{T}_5(Q_3(\mathbf{D})) = &\langle\mathcal{B}\!:\!2\rangle\langle\mathcal{A}\!:\!1\rangle\langle\mathcal{C}\!:\!1\rangle(\langle\mathcal{D}\!:\!2\rangle \cup \langle\mathcal{D}\!:\!3\rangle)\cup\\ &\langle\mathcal{B}\!:\!3\rangle(\langle\mathcal{A}\!:\!1\rangle \cup \langle\mathcal{A}\!:\!2\rangle)\times\\ &\times \big(\langle\mathcal{C}\!:\!1\rangle(\langle\mathcal{D}\!:\!2\rangle \cup \langle\mathcal{D}\!:\!3\rangle) \cup \langle\mathcal{C}\!:\!2\rangle\langle\mathcal{D}\!:\!3\rangle\big)\end{aligned}$$

where we have eliminated most $\times$ symbols for clarity.

Consider now the query $\pi_{A_R,B_R,B_S,D_T}Q_3$, a projected version of Q_3, and the computation of its result on the database $\mathbf{D}$ from Figure 8, factorised over the f-tree $\mathcal{T}_6$ shown in Figure 7. Assume that the algorithm **gen** uses the extension $\hat{\mathcal{T}}_6$ of $\mathcal{T}_6$. At node $\mathbf{D}$, we iterate over all ranges of tuples with the same D_T-values of T, but we still need to check whether the tuples in the newly-computed ranges satisfy the join condition $C_S = C_T$. By using the extension $\hat{\mathcal{T}}_6$ shown in Figure 7, the join condition is enforced by multiplying the $\mathbf{D}$-singletons with the result of **gen** at node $\mathcal{C}$. The resulting f-representation for the sample database $\mathbf{D}$ is

$$\begin{aligned}&\langle\mathcal{B}\!:\!2\rangle\langle\mathcal{A}\!:\!1\rangle(\langle\mathcal{D}\!:\!2\rangle\langle\rangle \cup \langle\mathcal{D}\!:\!3\rangle\langle\rangle)\cup\\ &\langle\mathcal{B}\!:\!3\rangle(\langle\mathcal{A}\!:\!1\rangle \cup \langle\mathcal{A}\!:\!2\rangle)\big(\langle\mathcal{D}\!:\!2\rangle\langle\rangle \cup \langle\mathcal{D}\!:\!3\rangle(\langle\rangle \cup \langle\rangle)\big),\end{aligned}$$

which gives $\mathcal{T}_6(Q_3(\mathbf{D}))$ after normalisation:

$$\begin{aligned}&\langle\mathcal{B}\!:\!2\rangle\langle\mathcal{A}\!:\!1\rangle(\langle\mathcal{D}\!:\!2\rangle \cup \langle\mathcal{D}\!:\!3\rangle)\cup\\ &\langle\mathcal{B}\!:\!3\rangle(\langle\mathcal{A}\!:\!1\rangle \cup \langle\mathcal{A}\!:\!2\rangle)(\langle\mathcal{D}\!:\!2\rangle \cup \langle\mathcal{D}\!:\!3\rangle)\end{aligned}$$

9. ANNOTATED DATABASES

In this section, we consider f-representations of provenance for conjunctive queries over annotated databases. Annotations are used to encode provenance information, conditions in uncertain tables, events in probabilistic databases, and tuple multiplicities to support bag semantics.

An annotated relation R is a relation with special *annotation attributes*, denoted by $\underline{R}$, which carry annotations in the form of tuple identifiers. Annotations are propagated through queries from input to the result [12, 8]; in particular, they cannot be projected away.

$\underline{R}$	A_R	B_R	$\underline{S}$	B_S	C_S	$\underline{T}$	C_T	D_T
r_{12}	1	2	s_{21}	2	1	t_{12}	1	2
r_{13}	1	3	s_{31}	3	1	t_{13}	1	3
r_{23}	2	3	s_{32}	3	2	t_{23}	2	3

Figure 9: An annotated database. The leftmost (underlined) column in each relation is the annotation attribute whose values are tuple identifiers.

Example 16. Figure 9 shows an annotated version of the database in Figure 8. Some tuples in the result of query Q_3 from Example 15 on this database are

$\underline{R}$	$\underline{S}$	$\underline{T}$	A_R	B_R	B_S	C_S	C_T	D_T
r_{12}	s_{21}	t_{12}	1	2	2	1	1	2
r_{12}	s_{21}	t_{13}	1	2	2	1	1	3
r_{13}	s_{31}	t_{12}	1	3	3	1	1	2
	$\dots$				$\dots$			

□

For an annotated tuple t in the query result, the annotation attributes hold the identifiers of the input tuples used to create t. Let us denote by $\mathcal{I}$ the annotation attributes of a relation.

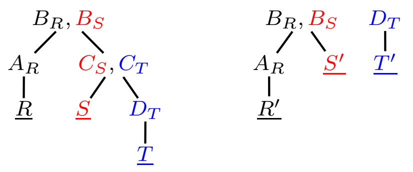

Figure 10: Left to right: f-tree $\mathcal{T}_5$ from Figure 7 extended with annotation attributes at leaves, and a provenance f-tree $\mathcal{T}_6$ for Q_4, which is a forest of two trees.

Definition 9. The *provenance* $\varphi(t)$ of a tuple t in the result of a query Q with projection list $\mathcal{P}$ on a database $\mathbf{D}$ is the relation $\pi_{\mathcal{I}}\sigma_{\bigwedge_{A\in\mathcal{P}} A=t(A)}Q(\mathbf{D})$. □

If we interpret relational product ($\times$) as multiplication and relational union ($\cup$) as addition, then $\varphi(t)$ becomes a *provenance polynomial* over the semiring of tuple identifiers with operations addition and multiplication [8]. Provenance polynomials admit different scenarios: in the Boolean semiring used in probabilistic databases and incomplete information, the identifiers are Boolean variables and the operations are logical "or" and "and"; in the semiring over natural numbers used for bag semantics, the identifiers are numbers and the operations are sum and product of numbers.

Example 17. The provenance of the tuple $\langle 1,2,2,1,1,2\rangle$ in the result of Q_3 given in Example 16 is the relation

$\underline{R}$	$\underline{S}$	$\underline{T}$
r_{12}	s_{21}	t_{12}

The provenance polynomial $\varphi(\langle\rangle)$ for the tuple $\langle\rangle$ in the result of the Boolean query $\pi_\emptyset Q_3$ is

$$r_{12}s_{21}t_{12} + r_{12}s_{21}t_{13} + r_{13}s_{31}t_{12} + r_{13}s_{31}t_{13} +$$
$$r_{13}s_{32}t_{23} + r_{23}s_{31}t_{12} + r_{23}s_{31}t_{13} + r_{23}s_{32}t_{23}.$$

9.1 Provenance Factorisation

The provenance of a result tuple can be factorised akin to a standard relation. An f-representation of the provenance $\varphi(\langle 1,2,2,1,1,2\rangle)$ in Example 17 is $\langle \underline{R}:r_{12}\rangle\langle \underline{S}:s_{21}\rangle\langle \underline{T}:t_{12}\rangle$, which is a product of *annotation singletons*.

The f-trees of the input query cannot be used to factorise effectively its provenance, unless the query is a product of relations. To see this, consider a Boolean query Q joining n relations. Any f-tree of Q has one node for each annotation attribute. These nodes are dependent, since any pair of annotation attributes are dependent (they belong to dependent relations). The f-trees are thus paths of length n and, according to Section 6.4, lead to poorly factorisable representations. Our factorisation approach for provenance of a query Q does not use f-trees of Q but of a query that can be derived from Q, as discussed next.

For a Boolean query Q, we take an f-tree $\hat{\mathcal{T}}$ of the equi-join $\hat{Q}$ of Q. This f-tree $\hat{\mathcal{T}}$ can be used to factorise the annotated result of $\hat{Q}$, i.e., the result together with its provenance. From the f-representation of the annotated query result, we can obtain an f-representation of the provenance of the result $\langle\rangle$ of Q by dropping all data singletons and keeping the annotation singletons only. This f-representation encodes exactly once each derivation of Q's result tuple $\langle\rangle$.

Example 18. Recall the Boolean query $\pi_\emptyset Q_3$ and the provenance $\varphi(\langle\rangle)$ whose polynomial is shown in Example 17. If we consider the f-tree $\mathcal{T}_5$ of the equi-join query Q_3, now extended with annotation attributes as shown in Figure 10, the f-representation of $\varphi(\langle\rangle)$ over this f-tree is

$$\langle r_{12}\rangle\langle s_{21}\rangle(\langle t_{12}\rangle \cup \langle t_{13}\rangle)\cup$$
$$(\langle r_{13}\rangle \cup \langle r_{23}\rangle)\big(\langle s_{31}\rangle(\langle t_{12}\rangle \cup \langle t_{13}\rangle) \cup \langle s_{32}\rangle\langle t_{23}\rangle\big).\square$$

For non-Boolean queries, we reduce the factorisation problem to that of a Boolean query. Indeed, given a query $Q = \pi_{\mathcal{P}}\sigma_\psi(R_1 \times \cdots \times R_n)$ on a database $\mathbf{D}$, the provenance of a result tuple $t \in Q(\mathbf{D})$ is the same as the provenance of the result of the Boolean query $Q' = \pi_\emptyset\sigma_{\psi'}(R'_1 \times \cdots \times R'_n)$ on a database $\mathbf{D}'$. The database $\mathbf{D}' = (\mathbf{R}'_1 \ldots, \mathbf{R}'_n)$ can be computed from the database $\mathbf{D}$ as follows. If a relation $\mathbf{R}_i$ with schema $\{A_1,\ldots,A_l\}$ has attributes $A_1,\ldots,A_k$ in $\mathcal{P}$ or equivalent to attributes in $\mathcal{P}$, then

$$\mathbf{R}'_i = \pi_{A_{k+1},\ldots,A_l}\sigma_{A_1=t(A_1)\wedge\cdots\wedge A_k=t(A_k)}(\mathbf{R}).$$

The condition ψ' is ψ restricted to the attributes of $\mathbf{D}'$. Different tuples in the result $Q(\mathbf{D})$ lead to the same Boolean query Q', yet to a different database $\mathbf{D}'$. We can now factorise the provenance of Q's result tuples using the approach for Boolean queries.

Example 19. Consider the query $Q_4 = \pi_{C_S}Q_3$ and the provenance $\varphi(\langle 2\rangle)$ of the result tuple $t = \langle 2\rangle$. We first rewrite the query Q_4 into a Boolean query Q'_4 by dropping the head attributes and their equivalence classes: $Q'_4 = \pi_\emptyset\sigma_{B_R=B_S}(R'\times S'\times T')$, where R' has schema $\{A_R, B_R\}$, S' has schema $\{B_S\}$ and T' has schema $\{D_T\}$. We also define the database $\mathbf{D}'$ to contain the annotated relations $\mathbf{R}' = \mathbf{R}$ from Figure 9, $\mathbf{S}' = \{\langle s_{32}, 3\rangle\}$ and $\mathbf{T}' = \{\langle t_{23}, 3\rangle\}$.

Now the provenance $\varphi(\langle 2\rangle)$ in $Q_4(\mathbf{D})$ is the same as the provenance $\varphi(\langle\rangle)$ in $Q'_4(\mathbf{D}')$:

$$\varphi(\langle 2\rangle) = \langle \underline{R}:r_{13}\rangle\langle \underline{S}:s_{32}\rangle\langle \underline{T}:t_{23}\rangle \cup \langle \underline{R}:r_{23}\rangle\langle \underline{S}:s_{32}\rangle\langle \underline{T}:t_{23}\rangle.$$

An f-tree $\mathcal{T}_6$ of the equi-join of Q'_4 is shown right in Figure 10. The f-representation of the annotated result $Q'_4(\mathbf{D}')$ over $\mathcal{T}_6$ is

$$\langle \mathcal{B}:3\rangle(\langle \mathcal{A}:1\rangle\langle \underline{R}:r_{13}\rangle \cup \langle \mathcal{A}:2\rangle\langle \underline{R}:r_{23}\rangle)\langle \underline{S}:s_{32}\rangle\langle \mathcal{D}:3\rangle\langle \underline{T}:t_{23}\rangle$$

and by dropping the data values we get the f-representation of $\varphi(\langle 2\rangle)$ over $\mathcal{T}_6$,

$$\mathcal{T}_6(\varphi(\langle 2\rangle)) = (\langle \underline{R}:r_{13}\rangle \cup \langle \underline{R}:r_{23}\rangle)\langle \underline{S}:s_{32}\rangle\langle \underline{T}:t_{23}\rangle.\square$$

The above two cases form a complete approach for factorising provenance of tuples in the results of an arbitrary conjunctive query Q. The f-trees that can be used for factorisation are valid for a specific query that is obtained from Q by first rewriting Q into a Boolean query, whereby the head attributes of Q are dropped, and then taking the equi-join of the latter. We call this query the *provenance query of* Q and its f-trees the *provenance f-trees of* Q. The provenance query of a given query Q can be computed in time linear in the size of Q and the size of the database schema. Optimal provenance f-trees can be found using the algorithm in Section 7. Given a provenance f-tree $\mathcal{T}$ of Q, the f-representation over $\mathcal{T}$ of the provenance $\varphi(t)$ of a tuple t in the result of Q is denoted by $\mathcal{T}(\varphi(t))$. Similar to f-representations of query results, the f-representation $\mathcal{T}(\varphi(t))$ is unique up to commutativity of product and union and can be computed using the algorithm in Section 8.

9.2 Readability of Provenance

Besides size, a further measure for provenance is readability. This measure has been previously defined for Boolean functions [5] and our motivation draws on the close connection between f-representations of annotated relations and algebraic factorisations of Boolean functions. Functions of low readability are preferred over functions of high readability, since the former can be implemented with smaller logical circuits. As discussed in the next section, our characterisation of queries by the readability of their results sheds light on a key structural property of the queries that goes beyond compactness of query results.

We next give a definition of readability tailored to f-representations.

Definition 10. An f-representation E is *read-k* if any value occurs in at most k singletons of E. The *readability* of a relation R is the smallest k such that R has a read-k f-representation. □

Example 20. The flat f-representation of provenance $\varphi(\langle\rangle)$, which is given in Example 17 as a polynomial, is read-4 since the value s_{31} occurs four times and no value occurs more than four times. Its f-representation over the f-tree $\mathcal{T}_5$ from Figure 10 is given in Example 18; it is read-2, since the values t_{12} and t_{13} occur twice and all other values only occur once. The readability of $\varphi(\langle\rangle)$ is 2 because it admits no read-1 f-representation. □

The main result of this section is the following characterisation of conjunctive queries based on the readability of provenance of their result tuples. For any query Q, let M be the maximum number of repeating relation symbols in Q.

Theorem 5. *For any conjunctive query Q, there is a rational number $r(Q)$ such that:*

- *For any database $\mathbf{D}$ and tuple $t \in Q(\mathbf{D})$, the readability of the provenance $\varphi(t)$ is at most $M \cdot |\mathbf{D}|^{r(Q)}$.*
- *For any provenance f-tree $\mathcal{T}$ of Q there exist arbitrarily large databases $\mathbf{D}$ and tuples $t \in Q(\mathbf{D})$ for which the f-representation $\mathcal{T}(\varphi(t))$ is at least read-$(|\mathbf{D}|/|Q|)^{r(Q)}$.*

The parameter $r(Q)$ is called the *readability width* of Q.

Our study of readability bounds follows the one on size bounds, with the difference that we now need only consider singletons for annotation attributes. The number of occurrences of any given singleton in any f-representation over an f-tree is quantified by Lemma 2. In particular, for a query result $Q(\mathbf{D})$ and an f-tree $\mathcal{T}$ of Q, an annotation attribute $\underline{R}$ and an identifier r of a tuple t, the number of occurrences of the singleton $\langle\underline{R}{:}r\rangle$ in $\mathcal{T}(Q(\mathbf{D}))$ is $|\pi_{\text{anc}(\underline{R})}\sigma_{\underline{R}=r}\mathcal{T}(Q(\mathbf{D}))|$. However, since the attribute $\underline{R}$ is a key for R, this is equal to $|\pi_{\text{anc}(\underline{R})}\sigma_{R=t}\mathcal{T}(Q(\mathbf{D}))|$, where the condition $R = t$ means that we assign to each attribute of R its value in tuple t. Just as the bounds for the number of singletons of type A were derived using the query Q_A, which is Q restricted to path(A), the number of individual annotation singletons $\langle\underline{R}{:}r\rangle$ are derived using a query Q_R: the query Q_R is Q restricted to path$(\underline{R})$ but *without* the attribute classes with attributes of R, since the attributes of R are already fixed by values in the tuple t. This is a key difference between our analysis of readability in case of annotated relations vs. size in case of standard relations.

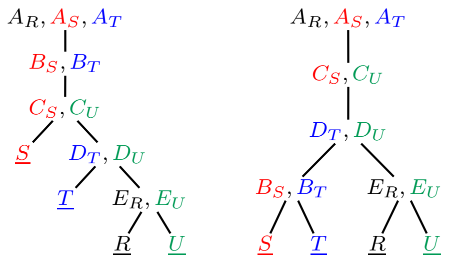

Figure 11: F-trees from Figure 3 extended with annotated attributes at leaves.

Lemma 9. *For any annotated query result $Q(\mathbf{D})$ and an f-tree $\mathcal{T}$ of Q, the number of occurrences of any $\underline{R}$-singleton in the f-representation $\mathcal{T}(Q(\mathbf{D}))$ is at most $|\mathbf{D}|^{\rho^*(Q_R)}$.*

Recall that the provenance $\varphi(t)$ of a tuple t in a query result $Q(\mathbf{D})$ can be factorised over any f-tree $\mathcal{T}$ of the provenance query $\tilde{Q}$ of Q. The f-representation $\mathcal{T}(\varphi(t))$ of $\varphi(t)$ is obtained by dropping all data singletons from the f-representation $\mathcal{T}(\tilde{Q}(\mathbf{D}'))$, where $\mathbf{D}'$ is constructed as described in Section 9.1. The number of occurrences of any annotation identifier in $\mathcal{T}(\varphi(t))$ is thus the same as in $\mathcal{T}(\tilde{Q}(\mathbf{D}'))$, for which we can use Lemma 9.

Similarly to $s(\mathcal{T})$, we define $r(\mathcal{T}) = \max_R \rho^*(\tilde{Q}_R)$ to be the maximal possible $\rho^*(\tilde{Q}_R)$ over all relations R from $\tilde{Q}$. This parameter controls the exponent of the readability in a similar way $s(\mathcal{T})$ does for sizes: for any database $\mathbf{D}$ and tuple $t \in Q(\mathbf{D})$, the f-representation $\mathcal{T}(\varphi(t))$ is at most read-$M \cdot |\mathbf{D}|^{r(\mathcal{T})}$. The factor M arises because a given value can appear in the singletons of up to M different types.

Finally, by defining $r(Q) = \min_{\mathcal{T}} r(\mathcal{T})$ to be the minimum possible $r(\mathcal{T})$ over all provenance f-trees $\mathcal{T}$ of Q, we obtain an upper bound for the readability of provenance of Q.

Corollary 6. *For any database $\mathbf{D}$ and tuple $t \in Q(\mathbf{D})$, the readability of the provenance $\varphi(t)$ is at most $M \cdot |\mathbf{D}|^{r(Q)}$.*

Example 21. Consider the Boolean query $\pi_\emptyset Q_3$ for Q_3 from Example 15. The f-tree $\mathcal{T}_5$ of Q_3 shown left in Figure 10 is a provenance f-tree of $\pi_\emptyset Q_3$. For relation R, all ancestor nodes of $\underline{R}$ contain attributes of R, so the query Q_R is empty and $\rho^*(Q_R) = 0$. This is also the case for the relation S. For relation T we have $Q_T = \sigma_{B_R = B_S}(\pi_{B_R} R \times \pi_{B_S} S)$. The hypergraph of Q_T has a single node and two edges of size 1, so $\rho^*(Q_T) = 1$. It follows that each annotation singleton of type $\underline{T}$ appears at most $|\mathbf{D}|$ times in any f-representation over $\mathcal{T}_5$, while all other annotation singletons appear at most once. The provenance of $\langle\rangle$ in $\pi_\emptyset Q_3(\mathbf{D})$ is therefore at most read-$|\mathbf{D}|$ for any database $\mathbf{D}$.

We now repeat Example 8, for readability bounds instead of size bounds.

Consider the Boolean query $\pi_\emptyset Q_2$ for Q_2 in Example 8, and the f-tree $\mathcal{T}_3$ of Q_2 extended with annotation attributes at leaves (shown left in Figure 11). The query Q_R has attributes $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$, its hypergraph is a triangle and hence $\rho^*(Q_R) = 3/2$. For the other relations, the fractional edge cover number of their queries is less and hence $r(\mathcal{T}_3) = 3/2$.

For the right f-tree $\mathcal{T}_4$ in Figure 11, each of Q_R, Q_S, Q_T and Q_U can be covered by a single relation and hence $r(\mathcal{T}_4) = 1$. This is the smallest possible value for an f-tree of Q_2, or equivalently, for a provenance f-tree of $\pi_\emptyset Q_2$, so $r(\pi_\emptyset Q_2) = 1$. We deduce that the provenance of $\langle\rangle$ in $\pi_\emptyset Q_2(\mathbf{D})$ is at most read-$|\mathbf{D}|$ for any database $\mathbf{D}$. $\square$

The lower bounds can also be adapted correspondingly.

LEMMA 10. *For any query Q, f-tree $\mathcal{T}$ of Q, and annotation attribute $\underline{R}$ in Q, there exist arbitrarily large databases $\mathbf{D}$ such that the number of occurrences of an $\underline{R}$-singleton in the f-representation $\mathcal{T}(Q(\mathbf{D}))$ of the annotated query result $Q(\mathbf{D})$ is at least $(|\mathbf{D}|/|Q|)^{\rho^*(Q_R)}$.*

Using the above lemma we obtain lower bounds on readability of f-representations with respect to f-trees.

COROLLARY 7. *For any provenance f-tree $\mathcal{T}$ of Q, there exist arbitrarily large databases $\mathbf{D}$ and tuple $t \in Q(\mathbf{D})$ such that the f-representation $\mathcal{T}(\varphi(t))$ is at least read-$(|\mathbf{D}|/|Q|)^{r(Q)}$.*

Theorem 5 follows from Corollaries 6 and 7.

10. DICHOTOMY FOR READABILITY

In case of conjunctive queries on annotated relations, a remarkable property holds: *hierarchical queries* [2] are precisely those queries for which the provenance of tuples in query results has bounded readability.

We next present this property in more detail. At the outset is the observation that there are queries for which the readability width $r(Q)$ is zero and hence the upper bound on readability is constant. Unlike the size parameter $s(Q)$, which is at least 1 for non-empty queries, $r(Q) = 0$ for a wide class of queries.

PROPOSITION 8. *A conjunctive query Q is hierarchical iff the readability width $r(Q)$ of Q is zero.*

For any non-hierarchical query Q, we have $r(Q) > 0$. However, $r(Q) = r(\mathcal{T}) = \rho^*(Q_R)$ for some f-tree $\mathcal{T}$ and relation R, so $r(Q) > 0$ implies $r(Q) \geq 1$. This creates a gap in the possible readability bounds for queries.

THEOREM 6. *Let Q be a conjunctive query.*
1. *If Q is hierarchical, then the readability of the provenance $\varphi(t)$ for any tuple $t \in Q(\mathbf{D})$ and database $\mathbf{D}$ is bounded by a constant.*
2. *If Q is non-hierarchical, for any provenance f-tree $\mathcal{T}$ of Q there exist arbitrarily large databases $\mathbf{D}$ and tuple $t \in Q(\mathbf{D})$ such that $\mathcal{T}(\varphi(t))$ is read-$\Omega(|\mathbf{D}|)$.*

For non-repeating queries, we can strengthen the above dichotomy to readability *irrespective* of f-trees. We first state the readability for the simplest non-hierarchical query:

$$Q_{nh} = \pi_\emptyset \sigma_{A_R=A_S \wedge B_S=B_T}(R \times S \times T)$$

where the relations R, S, T are over schemas $\{A_R\}$, $\{A_S, B_S\}$, and $\{B_T\}$ respectively. Consider the relation instances $\mathbf{R} = [N]$, $\mathbf{T} = [N]$ and $\mathbf{S} = [N] \times [N]$ annotated by identifiers r_i, t_j and s_{ij} respectively, with $1 \leq i, j \leq N$. The flat f-representation of Q_{nh}'s provenance is then

$$\Phi_N = \bigcup_{i,j=1}^{N} \langle r_i\rangle\langle s_{ij}\rangle\langle t_j\rangle.$$

LEMMA 11. *The relation Φ_N has readability $\frac{N}{2} + O(1)$.*

Lemma 11 can be generalised by having the relation T of size M, which may be different from N. The f-representation becomes $\bigcup_{i=1}^{N}\bigcup_{j=1}^{M} \langle r_i\rangle\langle s_{ij}\rangle\langle t_j\rangle$ and readability $\frac{NM}{N+M} + O(1)$.

The dichotomy result for non-repeating conjunctive queries is given next.

THEOREM 7. *Let Q be a non-repeating conjunctive query.*
1. *If Q is hierarchical, the readability of the provenance $\varphi(t)$ for any tuple $t \in Q(\mathbf{D})$ and database $\mathbf{D}$ is 1.*
2. *If Q is non-hierarchical, there exist arbitrarily large databases $\mathbf{D}$ and tuple $t \in Q(\mathbf{D})$ such that the readability of the provenance $\varphi(t)$ is $\Omega(\sqrt{|\mathbf{D}|})$.*

The hierarchical property plays a central role in studies with seemingly disparate focus, including the present one, probabilistic databases, parallel query evaluation, and streamed query evaluation. Our characterisation of query readability revolves around how far the query is from a hierarchical query. This is quantified by the readability width $r(Q)$ of the query, and is similar in spirit to existing width measures that capture the complexity of conjunctive queries, such as the fractional hypertree width [7, 14].

Theorem 7 draws on earlier work on probabilistic databases [15, 20], where read-once probabilistic events are useful since their exact probability can be computed in polynomial time. For read-m events with $m > 3$, probability computation is #P-hard [21]. In our case, however, a readability that is polynomial in the sizes of the input database and query is acceptable, since it means that the size of the f-representation of the query result is polynomial, too.

The hierarchical property also divides queries that can be evaluated in one step from those that cannot in the finite cursor machine model of computation [9]. In this model, queries are evaluated by first sorting each relation, followed by one pass over each relation. Furthermore, in the Massively Parallel computation model, any conjunctive query that can be evaluated (under bag semantics) with one synchronisation step is hierarchical [13].

11. EXTENSIONS AND FUTURE WORK

The results in this paper can be extended in several ways, e.g., by considering a larger query language and more expressive f-representations. We next highlight a few challenges, initial results, and point out open problems. We have compiled elsewhere a list of challenges in managing factorised representations of annotated relations and provenance polynomials [18].

11.1 Query Language Extensions

Of foremost importance is the extension of our factorisation framework to queries with aggregates and order-by clauses. The addition of selections with constants to our framework carries over immediately [17].

Beyond conjunctive queries, the readability results become more involved. We exemplify with simple in/dis-equality joins. Let $Q_N = \bigcup_{i,j=1;i\neq j}^{N} \langle r_i\rangle\langle s_j\rangle$ be the flat f-representation of the result of a disequality join $\sigma_{X_R \neq Y_S}(R \times S)$, where X_R and Y_S attributes of R and S respectively.

PROPOSITION 9. *The readability of Q_N is $\Omega(\frac{\log N}{\log\log N})$ and $O(\log N)$.*

If $i \neq j$ is replaced by $i \leq j$ in Q_N, the lower and upper bounds on readability still hold and we obtain an inequality query. In case Boolean factorisation is allowed, a lower bound of $\sqrt{\frac{\log N}{\log\log N}}$ on readability is already known [5].

11.2 Fractional Hypertree Decompositions

Earlier work on tractability of *Boolean* conjunctive queries has considered characterisations based on structural properties of the query hypergraph, such as hypertree width and its fractional version, e.g., [6, 14]. Queries of bounded width have polynomial combined complexity.

There is a strong connection between our work and this earlier work, as noted in Proposition 6: Each f-tree $\mathcal{T}$ of an equi-join query Q corresponds to a path decomposition H of Q's hypergraph. In addition, the parameter $s(\mathcal{T})$ is the fractional hyperpath width of H. This implies that the fractional hypertree width of any equi-join query Q is at most $s(Q)$. In Example 10 the query Q_n has fractional hypertree width 1 while $s(Q_n) = \Theta(\log n)$. However, fractional hypertree width determines the exponent for the complexity of answering the Boolean version of the query, while we represent the entire query result of size $O(|\mathbf{D}|^{\Theta(n)})$ in $O(|\mathbf{D}|^{\Theta(\log n)})$ time and space.

It is known that if the fractional hypertree width is w, it is possible to compute a decomposition of width $O(w^3)$ in polynomial time, but given a query Q and a number w, it is NP-hard to decide whether the fractional hypertree width of Q is less than w [14]. This hardness result is done by reduction from generalised hypertree width and is not useful in our case, since our f-trees are special cases of fractional hypertree decompositions. It remains open whether the decision problem $s(Q) \leq w$ for a given w is also NP-hard.

Also, finding f-representations with tight readability and size bounds with respect to general hypertree decompositions is a promising avenue for future research.

11.3 Queries on F-representations

Our factorisation framework can benefit query processing particularly when the input is given as an f-representation. We are building a database management system that presents relations at the logical layer and uses equivalent factorised representations at the physical layer. Such a system needs novel query evaluation and optimisation techniques.

The operators of physical query plans have to be adapted to work on factorised representations and unfold the factorisation to the extent needed to compute the f-representations of the query results. For instance, a join of two attributes needs unfolding unless the join attributes are along a same root-to-leaf path or siblings in the f-tree defining the input f-representation.

For query optimisation, specific rules that take the input f-tree into account are needed. For instance, there are several possibilities to change the input f-representation such that two join attributes become aligned on a path or siblings in the f-tree so as to facilitate efficient join evaluation.

Acknowledgments. Jakub Závodný was supported by an EPSRC DTA Grant EP/P505216/1.

12. REFERENCES

[1] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *Foundations of Computer Science (FOCS)*, 2008.

[2] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB Journal*, 16(4), 2007.

[3] K. M. Elbassioni, K. Makino, and I. Rauf. On the readability of monotone boolean formulae. In *International Computing and Combinatorics Conference (COCOON)*, 2009.

[4] M. C. Golumbic, A. Mintz, and U. Rotics. An improvement on the complexity of factoring read-once boolean functions. *Discrete Applied Mathematics*, 156(10), 2008.

[5] M. C. Golumbic, U. N. Peled, and U. Rotics. Chain graphs have unbounded readability. Technical report, University of Haifa, 2006.

[6] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. In *Principles of Database Systems (PODS)*, 1999.

[7] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *Journal of ACM*, 48, 2001.

[8] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Principles of Database Systems (PODS)*, 2007.

[9] M. Grohe, Y. Gurevich, D. Leinders, N. Schweikardt, J. Tyszkiewicz, and J. V. den Bussche. Database query processing using finite cursor machines. *Theory of Computing Systems*, 44(4), 2009.

[10] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *Symposium on Discrete Algorithms (SODA)*, 2006.

[11] M. Grohe, T. Schwentick, and L. Segoufin. When is the evaluation of conjunctive queries tractable? In *Symposium on Theory of Computing (STOC)*, 2001.

[12] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of ACM*, 31(4), 1984.

[13] P. Koutris and D. Suciu. Parallel evaluation of conjunctive queries. In *Principles of Database Systems (PODS)*, 2011.

[14] D. Marx. Approximating fractional hypertree width. In *Symposium on Discrete Algorithms (SODA)*, 2009.

[15] D. Olteanu and J. Huang. Using obdds for efficient query evaluation on probabilistic databases. In *Scalable Uncertainty Management (SUM)*, 2008.

[16] D. Olteanu, C. Koch, and L. Antova. World-set decompositions: Expressiveness and efficient algorithms. *Theoretical Computer Science*, 403(2-3), 2008.

[17] D. Olteanu and J. Závodný. Factorised representations of query results. Technical report, Oxford, April 2011. http://arxiv.org/abs/1104.0867.

[18] D. Olteanu and J. Závodný. On factorisation of provenance polynomials. In *Theory and Practice of Provenance (TaPP)*, 2011.

[19] N. Robertson and P. Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1), 1983.

[20] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[21] S. Vadhan. The Complexity of Counting in Sparse, Regular, and Planar Graphs. *SIAM Journal on Computing*, 32(2), 2001.