Size Bounds for Factorised Representations of Query Results

DAN OLTEANU and JAKUB ZÁVODNÝ, University of Oxford

We study two succinct representation systems for relational data based on relational algebra expressions with unions, Cartesian products, and singleton relations: f-representations, which employ algebraic factorisation using distributivity of product over union, and d-representations, which are f-representations where further succinctness is brought by explicit sharing of repeated subexpressions.

In particular we study such representations for results of conjunctive queries. We derive tight asymptotic bounds for representation sizes and present algorithms to compute representations within these bounds. We compare the succinctness of f-representations and d-representations for results of equi-join queries, and relate them to fractional edge covers and fractional hypertree decompositions of the query hypergraph.

Recent work showed that f-representations can significantly boost the performance of query evaluation in centralised and distributed settings and of machine learning tasks.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Relational databases, query processing*

General Terms: Theory, Algorithms, Design

Additional Key Words and Phrases: Succinct representation, data factorisation, conjunctive queries, size bounds, hypertree decompositions, query evaluation

ACM Reference Format:
Dan Olteanu and Jakub Závodný. 2015. Size bounds for factorised representations of query results. ACM Trans. Datab. Syst. 40, 1, Article 2 (March 2015), 44 pages.
DOI: http://dx.doi.org/10.1145/2656335

1. INTRODUCTION

Relational data is ubiquitous; methods for representing and storing relational data are therefore of great importance to database systems. Several storage approaches have been developed for relational data, including the standard row stores used by most traditional relational database systems, column stores [Batory 1979; Boncz et al. 1999; Stonebraker et al. 2005], approaches based on horizontal partitioning [Agrawal et al. 2004; Grund et al. 2010], and adaptive and declarative storage systems with a high-level interface for describing the physical representation of data [Cudré-Mauroux et al. 2009].

In this work, we study two succinct representation systems for relational data based on relational algebra expressions with unions, Cartesian products, and singleton relations (i.e., unary relations with one tuple): f-representations, which employ algebraic factorisation using distributivity of product over union, and d-representations, which are f-representations where further succinctness is brought by explicit sharing of repeated subexpressions. The relationship between a relation encoded as a set of tuples and an equivalent factorised representation is on par with the relationship between logic functions in disjunctive normal form and their equivalent nested formulas

J. Závodný was supported by an EPSRC DTA grant EP/P505216/1.
Authors' addresses: D. Olteanu and J. Závodný (corresponding author), University of Oxford, Wellington Square, Oxford OX1 2JD, UK; email: jakub.zavodny@oxfordalumni.org.

DOI: http://dx.doi.org/10.1145/2656335

obtained by algebraic factorisation. Similarly, the relationship between f-representations and d-representations is on par with that between formulas and circuits for logic functions.

Example 1.1. Consider the relation $R = \{(a,b,c) \in \mathbb{N}^3 : 1 \leq a < b < c \leq 5\}$ with tuples $(1,2,3)$, $(1,2,4)$, etc. If we write $\langle x \rangle$ for the singleton relation $\{x\}$, the tuples of R can be written as $\langle 1 \rangle \times \langle 2 \rangle \times \langle 3 \rangle$, $\langle 1 \rangle \times \langle 2 \rangle \times \langle 4 \rangle$, etc., and the relation R can be expressed by the flat relational algebra expression

$$\begin{aligned} R \;=\; & \langle 1 \rangle \times \langle 2 \rangle \times \langle 3 \rangle \;\cup\; \langle 1 \rangle \times \langle 2 \rangle \times \langle 4 \rangle \;\cup\; \langle 1 \rangle \times \langle 2 \rangle \times \langle 5 \rangle \;\cup\; \langle 1 \rangle \times \langle 3 \rangle \times \langle 4 \rangle \;\cup\; \langle 1 \rangle \times \langle 3 \rangle \times \langle 5 \rangle \\ & \cup\, \langle 1 \rangle \times \langle 4 \rangle \times \langle 5 \rangle \;\cup\; \langle 2 \rangle \times \langle 3 \rangle \times \langle 4 \rangle \;\cup\; \langle 2 \rangle \times \langle 3 \rangle \times \langle 5 \rangle \;\cup\; \langle 2 \rangle \times \langle 4 \rangle \times \langle 5 \rangle \;\cup\; \langle 3 \rangle \times \langle 4 \rangle \times \langle 5 \rangle. \end{aligned}$$

A more succinct factorised representation of R would be, for example,

$$R = \langle 1 \rangle \times \langle 2 \rangle \times (\langle 3 \rangle \cup \langle 4 \rangle \cup \langle 5 \rangle) \;\cup\; (\langle 1 \rangle \cup \langle 2 \rangle) \times \langle 3 \rangle \times (\langle 4 \rangle \cup \langle 5 \rangle) \;\cup\; (\langle 1 \rangle \cup \langle 2 \rangle \cup \langle 3 \rangle) \times \langle 4 \rangle \times \langle 5 \rangle.$$

Using d-representations, which use definitions to denote shared subexpressions, the representation could be further compacted to

$$\begin{aligned} X \;&:=\; \langle 1 \rangle \cup \langle 2 \rangle; \\ Y \;&:=\; \langle 4 \rangle \cup \langle 5 \rangle; \\ R \;&=\; \langle 1 \rangle \times \langle 2 \rangle \times (\langle 3 \rangle \cup Y) \;\cup\; X \times \langle 3 \rangle \times Y \;\cup\; (X \cup \langle 3 \rangle) \times \langle 4 \rangle \times \langle 5 \rangle. \end{aligned}$$

Both representation formalisms are complete for relational data in the sense that they can represent any relation instance. Moreover, they allow for fast retrieval of tuples of the represented relation: tuples can be enumerated with the same time complexity (constant per tuple) as listing them from the relation. Factorised representations can nevertheless be exponentially more succinct than traditional flat representations of relations as lists of tuples, such as in the presence of join dependencies and multi-valued dependencies in the relations. Results of conjunctive queries exhibit such dependencies and can be predictably factorised with an exponential succinctness gap when compared to flat relational representations.

In this article, we consider classes of f-representations and d-representations whose nesting structures are statically inferred from the query syntax. The nesting structures are defined by so-called f-trees for f-representations and d-trees for d-representations; they essentially exploit the join structure present in the query to achieve succinct representations of query results. Within these classes, we show how to define and effectively compute factorisations of asymptotically optimal size. Beyond these classes, seeking optimality becomes hard, for example, even finding large Cartesian products contained in a relation is NP-hard [Geerts et al. 2004] and for logical formulas the general problem of algebraic minimisation is known to be Σ_2^p-complete [Buchfuhrer and Umans 2008]. Furthermore, using f-trees and d-trees, factorisations of query results can be computed directly from the input database, without first computing the result in flat relational form.

Factorisations lie at the foundation of a new kind of data management system, with relations at the logical layer, and succinct factorised representations at the physical layer. They are of immediate relevance to practical settings where relations representing query results are used as input for subsequent processing, or shipped over the network in a distributed system. In such cases, a significant performance gain can be brought by the small size of factorised data when compared to its equivalent flat relational representation. We next exemplify with three distinct works that essentially rely on data factorisation to achieve scalability and improve performance.

The FDB centralised main-memory query engine supports queries with selections, projections, joins, aggregates, and ordering on f-representations [Bakibayev et al. 2012; 2013]. Experiments with FDB show that f-representations can boost the performance of

relational query processing by orders of magnitude in cases of flat input and factorised output or of query processing on factorised materialised views. The performance gap closely follows the size gap between flat and factorised representations for input and/or output data.

Recent work [Rendle 2013] applied factorisations to predictive modelling where feature vectors are computed by joining large tables. Whereas standard learning algorithms cannot scale to very large design matrices, exploiting repeating patterns in the design matrix as done by factorising equi-join results turns out to be key to scalability.

The F1 distributed database system that backs Google's AdWords business uses a restricted form of f-trees called hierarchical clustered schema [Shute et al. 2013]. In F1, scalability of both OLAP and OLTP workloads is mainly achieved by this factorisation, which increases data locality for common access patterns. The input tables are prejoined and clustered following the nesting structure of an f-tree defined by existing key-foreign-key constraints. The data is then partitioned across servers into factorisation fragments (called clusters) rooted at different tuples of the root table. Query processing is distributed, with intermediate results being shuffled between F1 nodes. We envisage further use of factorisations in the context of distributed database systems, where communication cost can be reduced by shipping small f-representations of intermediate query results. In contrast to generic compression techniques such as gzip, which are commonly used to mitigate network communication cost in distributed systems like F1, f-representations exploit the query structure to achieve arbitrarily better compression ratios while still preserving the relational nature of the compressed data and thus supporting queries in the compressed domain.

Factorised representations also naturally capture existing relational decompositions proposed in the literature: lossless decompositions defined by join dependencies, as investigated in the context of normal forms in database design [Abiteboul et al. 1995], conditional independence in Bayesian networks [Pearl 1989], factorisations of provenance polynomials of query results [Olteanu and Závodný 2011] used for efficient computation in probabilistic databases [Olteanu and Huang 2008; Sen et al. 2010], and Cartesian product decompositions of relations as studied in the context of incomplete information [Olteanu et al. 2007]. These existing decomposition techniques can be straightforwardly used to supply data in factorised form.

We study in this article the foundations of factorised representations, establishing the following properties.

—Factorised representations form a complete representation system for relational data. The tuples of a factorised representation, with or without definitions, can be enumerated with delay linear in the size of its schema and thus constant with respect to data complexity (Section 4).
—We introduce classes of factorised representations with the same nesting structures: these are so-called f-trees for f-representations, and d-trees for d-representations (Section 5). For a given conjunctive query, we can infer which f-trees and d-trees factorise all possible results of this query (Section 6).
—For any conjunctive query Q, there exist rational numbers $s(Q)$ and $s^{\uparrow}(Q)$ such that, for any input database $\mathbf{D}$, the result $Q(\mathbf{D})$ has an f-representation of size $O(|\mathbf{D}|^{s(Q)})$ and a d-representation of size $O(|\mathbf{D}|^{s^{\uparrow}(Q)})$. These bounds complement the known bound $O(|\mathbf{D}|^{\rho^*(Q)})$ for the size of the flat relational result $Q(\mathbf{D})$, where $\rho^*(Q)$ is the fractional edge cover of an equi-join query Q. Our size bounds are asymptotically optimal within the class of factorisations defined by f-trees and d-trees (Section 7).
—Factorised representations for results of conjunctive queries can be computed directly from the query and the input database. For equi-join queries we give

worst-case optimal algorithms: an f-representation of $Q(\mathbf{D})$ can be computed in time $O(|\mathbf{D}|^{s(Q)} \log |\mathbf{D}|)$ and a d-representation in time $O(|\mathbf{D}|^{s^\uparrow(Q)} \log |\mathbf{D}|)$ with respect to data complexity (Section 8).

—For results of equi-join queries, we quantify the succinctness gap between flat relations, f-representations, and d-representations, using the corresponding parameters $\rho^*(Q)$, $s(Q)$, and $s^\uparrow(Q)$. We show that $1 \leq s^\uparrow(Q) \leq s(Q) \leq \rho^*(Q) \leq |Q|$, where the factor between $s(Q)$ and $s^\uparrow(Q)$ is at most logarithmic in the size of the schema, while the factor between $\rho^*(Q)$ and $s(Q)$ can be as large as $|Q|$ (Section 9).

—Finally, factorisation of equi-join query results using f-trees and d-trees is closely related to path decompositions and tree decompositions of the query. We give a two-way translation between d-trees and tree decompositions showing that $s^\uparrow(Q)$ equals the fractional hypertree width of Q, and a one-way translation from f-trees to path decompositions showing that $s(Q)$ is at least the fractional hyperpath width of Q (Section 9).

To improve readability, the proofs of several formal statements in this article are deferred to the electronic appendix that can be accessed in the ACM Digital Library.

2. RELATED WORK

Factorised representations were originally introduced in Olteanu and Závodný [2012]. This article extends that landscape with factorised representations with definitions, provides full proofs for all claims, and places the factorisability parameters $s(Q)$ and $s^\uparrow(Q)$ within the picture of other known query parameters relevant to query decompositions. This work on factorisability of relations and query results lies at the foundation of a new kind of database systems that present relations at the logical layer and use equivalent but more succinct factorised representations at the physical layer. Recent work shows how to evaluate basic query operators, including selection, projection, join, aggregation, and ordering on factorised representations, and how such representations can boost the performance of query processing in relational databases in case of large input, intermediate, or final results [Bakibayev et al. 2012, 2013].

Equivalent to the special case of factorised representations over f-trees are *generalised hierarchical decompositions* (GHDs) and compacted relations over compaction formats. Existing work establishes the correspondences of GHDs to functional and multi-valued dependencies [Delobel 1978], and characterises selection conditions with disjunctions that can be performed on the compacted relations in one sequential pass [Bancilhon et al. 1982], but questions of succinctness have not been addressed. Nested and non-first normal form relations [Makinouchi 1977; Jaeschke and Schek 1982; Abiteboul and Bidoit 1986] are also structurally equivalent to factorised representations over f-trees, but are proposed as an alternative data model and not as a representation system for standard relation. Later work on nested relations [Ozsoyoglu and Yuan 1987] also considers the representation of a single flat relation by a nested relation, and infers possible nesting structures from join and multi-valued dependencies. This complements our results in Section 6, which characterise possible nesting structures for the results of a conjunctive query.

Various relational representation systems are subsumed by factorised representations of bounded depth. World-set decompositions in incomplete databases [Olteanu et al. 2007] and OR-objects that represent large spaces of possibilities or choices in design specification [Imielinski et al. 1991] are equivalent to products of unions of products of singletons. A polynomial-time factorisation algorithm has been proposed for decomposing a relation into a product of unions of products of singletons [Olteanu et al. 2007]. Products of unions of singletons are studied under different names (rectangles, bicliques in binary relations, n-sets, formal concepts) and several

representation systems are based on unions of products of unions of singletons, such as *generalised disjunctive normal forms* (GDNFs) studied as succinct presentations of inputs to CSPs [Chen and Grohe 2010], tilings of databases by bicliques, and n-sets or formal concepts [Geerts et al. 2004; Cerf et al. 2009]. The lazy, symbolic representation of the Cartesian product of two sets, which is used by factorised representations to avoid eager materialisation of all pairs of elements from the two sets, has also been recently used in the design of the GMP functional programming library for SQL-like processing on multisets [Henglein and Larsen 2010].

In relational databases, eliminating redundancy caused by join dependencies and multi-valued dependencies is traditionally addressed by normalising the relational schema [Kent 1983]. The trade-offs of using normalisation versus factorisation in relational database systems are discussed in recent work [Bakibayev et al. 2013]. Representation systems for relations based on join decompositions include minimal constraint networks [Gottlob 2012], but for these, data retrieval (tuple enumeration) is NP-hard. Tuple enumeration is constant time for acyclic queries [Bagan et al. 2007], in which case the input database together with the query already serve as a compact representation of the result. Decompositions of the query hypergraph, measuring the "degree of acyclicity" of the query, are traditionally used for classifying the tractability of Boolean queries and constraint satisfaction problems [Gottlob et al. 2000; Grohe and Marx 2006]. We draw a close connection of factorisations to hypertree decompositions in Section 9.

Representations utilising algebraic factorisation are not restricted to relational data. In the context of relational databases, factorisation can also be applied to provenance polynomials [Green et al. 2007] that describe how individual tuples of a query result depend on tuples of the input relations [Olteanu and Závodný 2011]. Algebraic and Boolean factorisations were considered for succinct representations of Boolean functions [Brayton 1987] and are closely related to binary decision diagrams, Boolean circuits, and other representations of Boolean functions.

3. 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 $\mathbf{D}$ is the sum of the sizes of its relations.

Queries. We consider conjunctive queries Q written in relational algebra form

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

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 $\mathcal{P}$ is a subset of $\bigcup_i \mathcal{S}_i$. Two attributes of Q are *equivalent* if they are transitively equal in the selection condition ψ. The equivalence class $\mathcal{A}$ of an attribute A is the set consisting of A and of all attributes equivalent to A. 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 called an equi-join query. The equi-join of Q is the query $\hat{Q} = \sigma_{\psi}(R_1 \times \cdots \times R_n)$. The *size* of Q is $|Q| = n$.

To simplify notation, we require that all relation symbols are distinct and have disjoint schemas. To capture queries with self-joins, we assume without loss of generality that mappings of relation symbols to database relations, as well as a correspondence between the attributes of relation symbols mapped to the same database relation, are

given together with the query.[1] The database $\mathbf{D}$ then only contains one relation instance for each set of relation symbols mapped to the same database relation. We thus consider queries with self-joins, though avoid the explicit use of aliases and renaming operators in the algebraic expressions.

The *hypergraph* of a query Q has one node for each attribute class of Q and one hyperedge containing all nodes with attributes of R for each relation symbol R. For any graph or hypergraph H, $V(H)$ denotes the set of nodes of H and $E(H)$ the set of edges of H.

Restrictions. For a conjunctive query $Q = \pi_{\mathcal{P}}(\sigma_{\psi}(R_1 \times \cdots \times R_n))$ and a set $S \subseteq \mathcal{P}$, the *S-restriction* of Q is the equi-join query $Q_S = \sigma_{\psi_S}(R_1^S \times \cdots \times R_n^S)$, where ψ_S and R_i^S are ψ and R_i, respectively, restricted to those attributes in the equivalence classes of attributes in S.

An S-restriction $\mathbf{D}_S$ of a database $\mathbf{D}$, with respect to the query Q, is constructed by projecting each relation of $\mathbf{D}$ onto those attributes in the equivalence classes of attributes in S. Following our simplification in case Q has self-joins, we first create a separate copy of each relation instance for each relation symbol referring to it before taking the projection. (Different relation symbols referring to the same relation instance may have different attributes equivalent to some attribute in S.)

Example 3.1. Consider relations R, S, and T over the schemas $\{A, B\}$, $\{B', C'\}$, and $\{C'', A''\}$ respectively. For a query $Q = \sigma_{B=B',C'=C'',A''=A}(R \times S \times T)$ and the set $X = \{A, B\}$, the X-restriction of Q is the query $Q_X = \sigma_{B=B',A''=A}(R^X \times S^X \times T^X)$, where R^X, S^X, and T^X are over schemas $\{A, B\}$, $\{B'\}$, and $\{A''\}$. For any database $\mathbf{D}$ with relations $\mathbf{R}$, $\mathbf{S}$, and $\mathbf{T}$, the X-restriction with respect to Q contains the relations $\mathbf{R}$, $\mathbf{S}^X = \pi_{B'}(\mathbf{S})$, and $\mathbf{T}^X = \pi_{A''}(\mathbf{T})$. The X-restriction of any projection query $\pi_{\mathcal{P}}Q$ is the same as the X-restriction of Q (as long as X is a subset of $\mathcal{P}$).

The S-restriction Q_S of Q only enforces the equality conditions from Q on attributes equivalent to S, and is in this sense less selective than Q. The result of Q_S on the restricted database $\mathbf{D}_S$ can thus contain more tuples than the projection $\pi_S(Q(\mathbf{D}))$.

PROPOSITION 3.2. *Let Q be a query, $\mathbf{D}$ be a database, and $Q_S, \mathbf{D}_S$ be their S-restrictions for a subset S of the head attributes of Q. Then $|\pi_S(Q(\mathbf{D}))| \leq |Q_S(\mathbf{D}_S)|$.*

Proposition 3.2 serves as a useful upper bound on the size of a projection by the size of an equi-join, and is used to establish our main upper bound result. See the electronic appendix for a detailed proof.

Dependencies. Factorisation of relations is possible whenever the values of two attributes are independent of each other, that is, knowing the value of one does not restrict the set of possible values of the other. Next we define the notion of independence of attributes and show that, in a query result, independence of attributes can be inferred statically from the query.

Two disjoint groups of attributes $\mathcal{A}$ and $\mathcal{B}$ of a relation $\mathbf{R}$ are called *independent conditioned on* another group of attributes $\mathcal{C}$, disjoint with $\mathcal{A}$ and $\mathcal{B}$, if $\mathbf{R}$ is a natural join $\mathbf{R}_A \bowtie_{\mathcal{C}} \mathbf{R}_B$ of two relations $\mathbf{R}_A$ and $\mathbf{R}_B$ with attributes including $\mathcal{A}$ and $\mathcal{B}$, respectively. $\mathcal{A}$ and $\mathcal{B}$ are *independent* if they are independent conditioned on the empty set. If two attributes are not conditionally independent, they are *dependent*.

Example 3.3. Consider the relation $\mathbf{R} = \{(a, b, c) : a \leq b, a \leq c\}$ over the schema $\{A, B, C\}$ and domain $\{1, 2\}$. No two attributes of R are independent because R cannot be

[1]The size of such mapping is at most linear in the size of the query and its schema, and hence does not impact the complexity analysis.

written as a product of two relations with nonempty schemas. However, in the relation $\pi_{\{B,C\}}\mathbf{R} = \{1,2\} \times \{1,2\}$, the attributes B and C are independent. In $\mathbf{R}$, attributes B and C are independent conditioned on A, because $\mathbf{R} = \mathbf{R}_B \bowtie_A \mathbf{R}_C$, where $\mathbf{R}_B = \{(a,b) : a \leq b\}$ and $\mathbf{R}_C = \{(a,c) : a \leq c\}$. Finally, attributes A and B are dependent in $\mathbf{R}$, since $\mathbf{R}$ cannot be written as a join of two relations on C.

In case the relation $\mathbf{R}$ is the result of a conjunctive query Q, we can deduce specific dependency information by static analysis of Q. For a query Q, two head attributes A and B are *Q-dependent* if any of the following statements hold:

—they belong to the same relation in Q;
—there is a chain of relations $R_1, \ldots, R_k$ in Q such that A is in the schema of R_1, B is in the schema of R_k, and each successive R_i and R_{i+1} are joined on an attribute that does not belong to the projection list $\mathcal{P}$ and neither does any equivalent attribute; and
—A is equivalent to A' and B is equivalent to B', where A' and B' are Q-dependent.

PROPOSITION 3.4. *Two attributes A and B are Q-dependent for a query Q if and only if there exists a database $\mathbf{D}$ for which A and B are dependent in the relation $Q(\mathbf{D})$.*

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.

4. FACTORISED REPRESENTATIONS

In this section we introduce the central concepts studied in this work: two succinct representation systems for relational data. The basic idea is to represent relations symbolically as expressions in a fragment of relational algebra consisting of union, Cartesian product, and so-called singleton relations, which are unary relations with one tuple. We call such representations *factorised representations* or *f-representations*, since they employ algebraic factorisation to nest products and unions and hence express combinations of values symbolically. Further succinctness can be achieved by introducing symbolic references into the representations, so that repeated subexpressions can be defined only once and referred to several times. Factorised representations with definitions are called *d-representations*.

4.1. Factorised Representations

Factorised representations of relations are defined as typed relational algebra expressions consisting of unions, Cartesian products, and singleton relations [Olteanu and Závodný 2012].

Definition 4.1. A *factorised representation*, or f-representation for short, over a schema $\mathcal{S}$ is a relational algebra expression of one of the following forms:

—$\emptyset$, representing the empty relation over schema $\mathcal{S}$;
—$\langle\rangle$, representing the relation consisting of the nullary tuple, if $\mathcal{S} = \emptyset$;
—$\langle A{:}a\rangle$, representing the unary relation with a single tuple with value a, if $\mathcal{S} = \{A\}$ and a is a value in the domain $\mathcal{D}$;
—$(E_1 \cup \cdots \cup E_n)$, representing the union of the relations represented by E_i, where each E_i is an f-representation over $\mathcal{S}$; or
—$(E_1 \times \cdots \times E_n)$, representing the Cartesian product of the relations represented by E_i, where each E_i is an f-representation over some schema $\mathcal{S}_i$ such that $\mathcal{S}$ is the disjoint union of all $\mathcal{S}_i$.

The expressions $\langle A{:}a\rangle$ are called *singletons of type A* (or A-singletons for short) and the expression $\langle\rangle$ is called the nullary singleton.

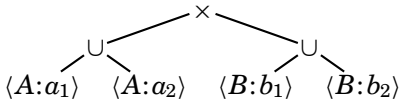
We write $[\![E]\!]$ for the relation represented by an f-representation E. Different f-representations can represent the same relation, two f-representations E_1 and E_2 over the same schema are *equivalent* if $[\![E_1]\!] = [\![E_2]\!]$.

Example 4.2. The f-representation $(\langle A{:}a_1\rangle \cup \langle A{:}a_2\rangle) \times (\langle B{:}b_1\rangle \cup \langle B{:}b_2\rangle)$ over the schema $\{A, B\}$ represents the relation $\{(a_1, b_1), (a_1, b_2), (a_2, b_1), (a_2, b_2)\}$ over $\{A, B\}$.

Any relation has at least one f-representation, the so-called *flat* f-representation that is a (possibly empty) union of products of singletons, where each product of singletons represents a distinct tuple in the relation. This property of the representation system is called completeness; factorised representations are hence a complete representation system for relational data.

Any f-representation has a parse tree whose internal nodes are unions and products, and whose leaves are singletons or empty relations. Relational algebra expressions and their parse trees are equivalent ways of describing f-representations; we use them interchangeably in this article. Relational algebra expressions are better suited for readability in text; their parse trees are better suited for formal analysis, proofs, and algorithms. Parentheses in relational expressions are omitted when this helps clarity.

Example 4.3. The f-representation $(\langle A{:}a_1\rangle \cup \langle A{:}a_2\rangle) \times (\langle B{:}b_1\rangle \cup \langle B{:}b_2\rangle)$ has the following parse tree.



4.2. Factorised Representations with Definitions

We now introduce the representation system of factorised representations with definitions, called d-representations. While f-representations eliminate redundancy in relations by expressing products of unions of expressions symbolically instead of a union of products of expressions, they may still contain multiple copies of the same expression that cannot be removed by further factorisation. This redundancy can be eliminated by defining (and physically storing) the subexpression only once and referring to this definition (using a pointer to the single stored copy) at each of its occurrences in the representation.

Definition 4.4. A *factorised representation with definitions*, or d-representation for short, is a set of named expressions $\{N_1 := D_1, \ldots, N_n := D_n\}$, where each name N_i is a unique symbol and each D_i is an expression with products, unions, singletons, and names of other expressions. Formally, an expression over a schema $\mathcal{S}$ is of one of the following:

—$\emptyset$, representing the empty relation over $\mathcal{S}$;
—$\langle\rangle$, representing the relation consisting of the nullary tuple, if $\mathcal{S} = \emptyset$;
—$\langle A{:}a\rangle$, representing the unary relation with a single tuple with value a, if $\mathcal{S} = \{A\}$ and a is a value in the domain $\mathcal{D}$;
—$(E_1 \cup \cdots \cup E_n)$, representing the union of the relations represented by E_i, where each E_i is an expression over $\mathcal{S}$;
—$(E_1 \times \cdots \times E_n)$, representing the Cartesian product of the relations represented by E_i, where each E_i is an expression over schema $\mathcal{S}_i$ such that $\mathcal{S}$ is the disjoint union of all $\mathcal{S}_i$; or
—a name N_i of another expression D_i over $\mathcal{S}$, representing the same relation as D_i.

The schema of a d-representation is the schema of its first expression D_1. We require that each D_i only contains names N_j with $j > i$, and that each name N_j with $j > 1$ is

used at least once. In this article, unless explicitly defined otherwise, the name of an expression D_i will always be ${}^{\uparrow}D_i$ (read: a pointer to D_i).

Any d-representation D over a schema $\mathcal{S}$ represents a relation $[\![D]\!]$ over $\mathcal{S}$. For any d-representation D consisting of expressions $D_1, \ldots, D_n$, we can start with the root expression D_1 and repeatedly replace the names ${}^{\uparrow}D_j$ by the expressions D_j until we obtain a single expression without names, that is, an f-representation. This f-representation is called the *traversal* of D, and D represents the same relation as its traversal.

Any f-representation E can be identified with the d-representation $\{E\}$: E is the traversal of $\{E\}$ and they represent the same relation. Therefore, d-representations also form a complete representation system for relational data.

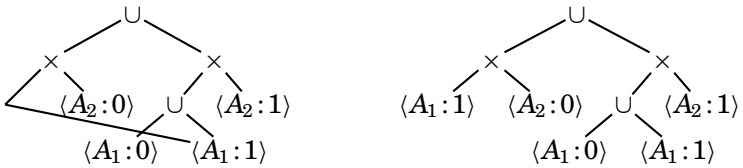
Just as any f-representation has a parse tree, any d-representation has a directed acyclic parse graph which can be constructed as a collection of the parse trees of the constituent expressions, in which any leaf corresponding to a reference ${}^{\uparrow}D_j$ is identified with the root of the parse tree of expression D_j. The traversal of a d-representation viewed as a parse graph is constructed by listing its nodes in depth-first order without marking them as visited (thus possibly listing some nodes multiple times). More precisely, for any node N with edges to nodes $C_1, \ldots, C_n$, let traversal(N) be a tree with a copy of N as a root and traversal(C_1), ..., traversal(C_n) as children subtrees; then *traversal* of a d-representation D is traversal(root(D)).

Similar to f-representations, we use relational expressions with definitions and parse graphs as interchangeable ways to describe d-representations.

Example 4.5. Consider the relation $\mathbf{R}_n$ over schema $\{A_1, \ldots, A_n\}$ whose tuples are all binary sequences $(a_1, \ldots, a_n)$ with no two consecutive zeros. The d-representation consisting of

$$D_{1,0} = \langle A_1 : 0\rangle, \quad D_{1,1} = \langle A_1 : 1\rangle \qquad \text{and}$$
$$D_{k,0} = {}^{\uparrow}D_{k-1,1} \times \langle A_k : 0\rangle, \quad D_{k,1} = ({}^{\uparrow}D_{k-1,0} \cup {}^{\uparrow}D_{k-1,1}) \times \langle A_k : 1\rangle \qquad \text{for } k = 2, \ldots, n,$$

and root $D = {}^{\uparrow}D_{n,0} \cup {}^{\uparrow}D_{n,1}$, represents the relation $\mathbf{R}_n$.[2] This can be seen by showing inductively over k that $D_{k,d}$ represents the relation $\sigma_{A_k=d}(\mathbf{R}_k)$. Depicted next is the parse graph of D, and the parse tree of its traversal, for the case $n = 2$.



Since f-representations are d-representations, all definitions and results for arbitrary d-representations mentioned in the sequel also apply to f-representations. We next introduce several notions that are used later.

Definition 4.6. A d-representation is *normal* if:

—it contains no empty relation or nullary singleton as a subexpression, unless it is itself the empty relation or the nullary singleton;
—all its products are at least binary; and
—no child of a union is a union.

[2]In Definition 4.4 the expressions D_i of a d-representation are indexed by natural numbers $i = 1, \ldots, n$, but any partial order with a least element is sufficient and can be re-indexed by consecutive naturals.

The first condition in Definition 4.6 ensures that an expansion of a normal d-representation into a flat representation is a union of well-formed tuples.

Definition 4.7. By expanding a (nonempty, nonnullary) normal f-representation using the distributivity of product over union, we obtain an equivalent flat f-representation that is a union of products of non-nullary singletons, which we call *monomials*. The monomials of a normal d-representation are those of its traversal. A normal d-representation is *deterministic* if its monomials are all distinct.

The monomials of a normal d-representation correspond to those tuples of the relation obtained by interpreting the d-representation under bag semantics. For a deterministic d-representation D the monomials are all distinct and hence also correspond to the tuples of its relation $[\![D]\!]$ under set semantics. In Section 4.4 we show that the tuples of a normal and deterministic d-representation can be enumerated with time delay proportional to the tuple size and independent of the number of tuples.

4.3. Representation Size

The size of a d-representation is determined by the total length of the expressions, when stored as a set of factorised expressions, and by the number of its nodes plus the number of its edges, when stored as a parse graph. Both measures are within a constant factor of each other; we will next use the former one. A further size measure is the number of singleton nodes in the representation.

Definition 4.8. The size $|E|$ of a d-representation E is the total number of its singletons, empty set symbols, unions, products, and occurrences of expression names. The number of singletons in E is denoted by $\|E\|$.

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 4.9. 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 $2n$ singletons, while any equivalent flat f-representation has $n \cdot 2^n$ singletons.

By deduplicating common subexpressions, d-representations can represent relations even more succinctly than their traversal f-representations. This size reduction is similar in spirit to the reduction in representation size for Boolean circuits when compared to equivalent Boolean formulas.

Example 4.10. The d-representation D from Example 4.5 has size $O(n)$, while the size of its traversal is exponential in n since only the singleton $\langle A_1:1\rangle$ occurs F_n times (F_n denotes the n^{th} Fibonacci number).

We will further study representation succinctness in Sections 7 and 9.

4.4. Constant-delay Enumeration of Encoded Records

Examples 4.9 and 4.10 show that f-representations and d-representations can be exponentially smaller than the relations they represent. The records of a relation encoded as a normal deterministic d-representation, or its f-representation traversal, can nevertheless be enumerated with the same complexity as listing them from the relation.

THEOREM 4.11. *The tuples of a normal deterministic d-representation D over a schema $\mathcal{S}$ can be enumerated with $O(|\mathcal{S}|)$ delay and space.*

PROOF. We assume that the d-representation is stored as a parse graph, but any representation allowing constant-time enumeration of elements of unions and products is sufficient.

Enumeration algorithm. We explore the d-representation D using depth-first search. For each union, we follow the edge to its first child, and construct a list $\mathcal{L}$ of nodes visited in pre-order. We then repeat the following.

(1) We output the product of all singletons in $\mathcal{L}$.
(2) We find the last union node U in the list $\mathcal{L}$ for which its child C in $\mathcal{L}$ is not its last child. If such U exists, we remove all nodes after C from $\mathcal{L}$ and replace C by the next child of U. If such U does not exist, we terminate.
(3) We explore D using depth-first search. For each union in $\mathcal{L}$, we follow the edge to its child in $\mathcal{L}$ and, for each union not in $\mathcal{L}$, we follow the edge to its first child. We update the list $\mathcal{L}$ to a list of nodes visited in pre-order during the search.

Termination. If we order the nodes of traversal(D) in pre-order, by each repetition of step (3) the list $\mathcal{L}$ becomes lexicographically greater. Since there are finitely many possible lists $\mathcal{L}$, the algorithm terminates.

Correctness. The tuples of a deterministic d-representation are the same as its monomials. Each monomial of a given normal d-representation D is a product of the singletons reached by recursively exploring D, choosing one child at each union, and all children at each product. Any choice of children at the unions of D corresponds to a monomial of D, therefore each product output in step (1) is a monomial of D. Conversely, for any monomial m of D, consider the choice of children at each union that generates m, and let $\mathcal{L}_m$ be the pre-ordered list of nodes visited by a depth-first search of D that only follows the chosen children at each union. During the execution of the enumeration algorithm, the first union of $\mathcal{L}$ will cycle through its children, so eventually the child of the first union in $\mathcal{L}$ will be identical to $\mathcal{L}_m$. The first time this happens, all other unions in $\mathcal{L}_m$ will have their first child in $\mathcal{L}_m$. The next union in $\mathcal{L}$ is then the same as in $\mathcal{L}_m$, and at some point its child in $\mathcal{L}$ will be identical to $\mathcal{L}_m$. By induction we can show that, at some point, all unions in $\mathcal{L}$ will have the same children in $\mathcal{L}$ as in $\mathcal{L}_m$, hence $\mathcal{L}$ and $\mathcal{L}_m$ will be identical and m will be output in the next execution of step (1). Moreover, since $\mathcal{L}$ strictly increases under lexicographic order, each monomial is only output once.

Delay and Space. For any choice of children at the unions, we reach exactly $|\mathcal{S}|$ singletons; one for each attribute. Since the products are at least binary, we reach at most $|\mathcal{S}| - 1$ of them and, since there are no directly nested unions, the number of reached unions is $O(|\mathcal{S}|)$. The size of $\mathcal{L}$ is therefore $O(|\mathcal{S}|)$, and steps (1) and (2) of the algorithm take time $O(|\mathcal{S}|)$. The initial depth-first search takes time linear in the number of explored nodes, which is $O(|\mathcal{S}|)$. The same holds for the depth-first search in step (3). Its choices of children at unions are at first dictated by the list of nodes $\mathcal{L}$, but the nodes in $\mathcal{L}$ are listed in pre-order, so each can be accessed in constant time during the search. □

Note that $O(|\mathcal{S}|)$ delay is optimal since each tuple has size $O(|\mathcal{S}|)$; the same time delay is achieved when listing the tuples from a readily materialised list. With respect to data complexity (where the schema size is constant), we thus enumerate the tuples with constant delay and space.

5. F-TREES AND D-TREES

In this section we introduce classes of normal f-representations and d-representations with uniform nesting structures, called f-trees and d-trees, respectively. Similar to relational schemas, f-trees and d-trees specify the set of attributes of the represented relation. In addition, they encode structural information that acts as a data-independent

factorisation pattern for f-representations and d-representations. In the next sections, we define the space of possible f-trees and d-trees for any result of a given query Q and show how to efficiently find representations that have optimal sizes within such classes. While size optimality of succinct representations is in general hard to achieve as it draws back to minimality of logic functions using algebraic factorisation [Brayton 1987], we are able to give such optimality results for representations within the classes of f-trees and d-trees, where the expensive computation depends only on the size of these small class descriptions and not on the size of the data. Furthermore, recent work showed that f-trees can effectively guide query processing on f-representations as they give a measure for how expensive structural transformations of representations can be [Bakibayev et al. 2012, 2013].

We next give exact conditions under which a relation admits an f-representation over a given f-tree (or d-representation over a given d-tree) and precisely characterise such representations if they exist. We also give algorithms to compute these representations in quasilinear time complexity.

5.1. F-Trees for F-Representations

We first introduce f-trees, which define the schemas as well as the nesting structures of f-representations.

Definition 5.1. An *f-tree* over a schema $\mathcal{S}$ is a rooted forest with each node labelled by a nonempty subset of $\mathcal{S}$ such that each attribute of $\mathcal{S}$ occurs in exactly one node.

An f-tree dictates the nesting structure of an f-representation: the shape of the f-tree specifies a hierarchy of attributes by which we group the tuples of the represented relation in the f-representation. We group the tuples of the relation by the values of the attributes labelling the root, factor out the common value in each group, and then continue recursively on each group using the attributes lower in the f-tree. Branching into several subtrees denotes (conditional) independence of attributes in the different subtrees; this leads to a product of f-representations over the individual subtrees. For each node, all attributes labelling the node have equal values in the represented relation.

Definition 5.2. We say that an f-representation E is over a given f-tree $\mathcal{T}$ if it satisfies the following.

—If $\mathcal{T}$ is empty, then $E = \emptyset$ or $E = \langle\rangle$.
—If $\mathcal{T}$ is a single node labelled by $\{A_1, \ldots, A_k\}$, then

$$E = \bigcup_a \langle A_1 : a\rangle \times \cdots \times \langle A_k : a\rangle,$$

where the union $\bigcup_a$ is over a collection of distinct values a.
—If $\mathcal{T}$ is a single tree with a root labelled by $\{A_1, \ldots, A_k\}$ and a nonempty forest $\mathcal{U}$ of children, then

$$E = \bigcup_a \langle A_1 : a\rangle \times \cdots \times \langle A_k : a\rangle \times E_a,$$

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 forest of trees $\mathcal{T}_1, \ldots, \mathcal{T}_k$, then

$$E = E_1 \times \cdots \times E_k,$$

where each E_i is an f-representation over $\mathcal{T}_i$.

Example 5.3. Consider a relation with schema $\{A, B, C\}$ and domain $\mathcal{D} = \{1, \ldots, 5\}$ that represents the inequalities $A < B < C$, as in Example 1.1. An f-representation of

this relation over the following f-tree is given next.

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

Example 5.4. An f-representation over a disconnected f-tree (forest) is a product decomposition of the represented relation. For example, the f-representation $(\langle A{:}a_1\rangle \cup \langle A{:}a_2\rangle) \times (\langle B{:}b_1\rangle \cup \langle B{:}b_2\rangle)$ from Example 4.2 is over the f-tree consisting of two disconnected root nodes A and B.

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. However, in case an f-representation over a given f-tree exists, it is unique up to the commutativity of product and union, and we characterise it precisely in the following section.

Example 5.5. 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 5.3, 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.

PROPOSITION 5.6. *Any f-representation over an f-tree is normal and deterministic.*

PROOF. The normality condition is syntactic and easy to prove: from Definition 5.2 of an f-representation E over an f-tree $\mathcal{T}$, E contains no $\emptyset$ or $\langle\rangle$ unless it is $\emptyset$ or $\langle\rangle$ itself, there are no directly nested unions, and all expressions in Definition 5.2 can be parsed so that all products have at least two arguments.

Determinism can be proven by bottom-up induction over $\mathcal{T}$. For any node $\mathcal{A}$, any f-representation over $\mathcal{A}$ is deterministic as it is a union of $\langle\mathcal{A}{:}a\rangle$ for distinct values a. For a single tree $\mathcal{T}$ with root $\mathcal{A}$ and forest of children $\mathcal{U}$, any f-representation E_a over $\mathcal{U}$ is deterministic by the induction hypothesis, and hence the f-representations $\langle\mathcal{A}{:}a\rangle \times E_a$ are all deterministic and disjoint for different a. Hence any f-representation over $\mathcal{T}$ of the form $\bigcup_a \langle\mathcal{A}{:}a\rangle \times E_a$ is also deterministic. Finally, any f-representation over a forest $\mathcal{T}_1, \ldots, \mathcal{T}_k$ is a product of f-representations over the $\mathcal{T}_i$. Each of these is deterministic by the induction hypothesis, and the product of deterministic f-representations over disjoint schemas is deterministic. □

We next introduce notations concerning f-trees. For any node $\mathcal{A}$ of an f-tree $\mathcal{T}$, $\mathcal{T}_\mathcal{A}$ denotes the subtree of $\mathcal{T}$ rooted at $\mathcal{A}$. By a *subtree* of $\mathcal{T}$ we mean a subtree of the form $\mathcal{T}_\mathcal{A}$ for some $\mathcal{A}$. By a *forest* of $\mathcal{T}$ we mean a set $\{\mathcal{T}_\mathcal{B}\}$ of all children $\mathcal{B}$ of some node in $\mathcal{T}$ or of all roots $\mathcal{B}$ of $\mathcal{T}$. We denote by $\mathrm{anc}(\mathcal{A})$ the set of all attributes at nodes that are ancestors of $\mathcal{A}$ in $\mathcal{T}$, and by $\mathrm{path}(\mathcal{A})$ the set of attributes at the ancestors of $\mathcal{A}$ and at $\mathcal{A}$. We overload the function anc to also retrieve the set of ancestor attributes of attributes, subtrees, and forests. The node containing an attribute A or attributes $\{A_i\}$ is denoted by $\mathcal{A}$ and we speak interchangeably of an f-tree node and of its set of attributes. For any node $\mathcal{A} = \{A_1, \ldots, A_k\}$, we use the shorthand $\langle\mathcal{A}{:}a\rangle$ for the product $\langle A_1{:}a\rangle \times \cdots \times \langle A_k{:}a\rangle$. Finally, we use shorthands such as $\pi_\mathcal{A}$, $\pi_{\mathcal{T}_\mathcal{A}}$ or $\pi_\mathcal{U}$ to mean the projection on the attributes of a node $\mathcal{A}$, a subtree $\mathcal{T}_\mathcal{A}$, or a forest $\mathcal{U}$, and $\mathrm{anc}(\mathcal{A}) = t$ to mean the selection condition $\wedge_{B\in\mathrm{anc}(\mathcal{A})} B = t(B)$, which enforces that tuples agree with t on the attributes $\mathrm{anc}(\mathcal{A})$.

Example 5.7. 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) $= \mathcal{A} \cup \mathcal{B} \cup \mathcal{C} = \{A_R, A_S, A_T, B_R, B_S, C\}$. The tree $\mathcal{T}_\mathcal{B}$ has root $\mathcal{B} = \{B_R, B_S\}$ and children $\mathcal{C} = \{C\}$ and $\mathcal{D} = \{D\}$.

5.2. Constructive Definition of F-Representations over F-Trees

For any relation $\mathbf{R}$ and any f-tree $\mathcal{T}$ over the same schema, we now explicitly define an f-representation $\mathcal{T}(\mathbf{R})$. We then prove that $\mathcal{T}(\mathbf{R})$ is the unique f-representation of $\mathbf{R}$ over $\mathcal{T}$, if one exists. The constructive characterisation of $\mathcal{T}(\mathbf{R})$ is used later in this article to reason about the representability of query results by f-representations, to construct f-representations algorithmically, and to derive bounds on their size.

As the basic building block of $\mathcal{T}(\mathbf{R})$, we first define f-representations $E(\mathbf{R}, \mathcal{X}, t)$ over subtrees or forests $\mathcal{X}$ of $\mathcal{T}$ and (context) tuples t over attributes anc($\mathcal{X}$). They are computed from relations $\pi_\mathcal{X}\sigma_{\text{anc}(\mathcal{X})=t}\mathbf{R}$, which are vertical-horizontal partitions of $\mathbf{R}$, but may not necessarily represent them exactly. We then specify conditions under which each $E(\mathbf{R}, \mathcal{X}, t)$ indeed represents the relation $\pi_\mathcal{X}\sigma_{\text{anc}(\mathcal{X})=t}\mathbf{R}$ and show how their composition into the f-representation $\mathcal{T}(\mathbf{R})$ represents the relation $\mathbf{R}$.

Definition 5.8. Let $\mathcal{T}$ be an f-tree and $\mathbf{R}$ a relation over the same schema. Let $\mathcal{T}(\mathbf{R})$ be the expression $E(\mathbf{R}, \mathcal{T}, \langle\rangle)$, where, for any subtree or forest $\mathcal{X}$ in $\mathcal{T}$, and any tuple t over anc($\mathcal{X}$), the expression $E(\mathbf{R}, \mathcal{X}, t)$ is defined recursively as follows.

—For any leaf $\mathcal{A}$,

$$E(\mathbf{R}, \mathcal{T}_\mathcal{A}, t) = \bigcup_{a\in A}\langle\mathcal{A}{:}a\rangle,$$

where $A = \pi_{A_1}\sigma_{\text{anc}(\mathcal{A})=t}\mathbf{R}$ and $A_1 \in \mathcal{A}$.

—For any subtree $\mathcal{T}_\mathcal{A}$ with root $\mathcal{A}$ and children $\mathcal{T}_1, \ldots, \mathcal{T}_k$,

$$E(\mathbf{R}, \mathcal{T}_\mathcal{A}, t) = \bigcup_{a\in A}\langle\mathcal{A}{:}a\rangle \times E(\mathbf{R}, \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}, t \times \langle\mathcal{A}{:}a\rangle),$$

where $A = \pi_{A_1}\sigma_{\text{anc}(\mathcal{A})=t}\mathbf{R}$ and $A_1 \in \mathcal{A}$.

—For each nonempty forest $\{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$,

$$E(\mathbf{R}, \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}, t) = E(\mathbf{R}, \mathcal{T}_1, t) \times \cdots \times E(\mathbf{R}, \mathcal{T}_k, t).$$

—For $\mathcal{T}$ empty, $E(\mathbf{R}, \mathcal{T}, \langle\rangle)$ is $\emptyset$ if $\mathbf{R} = \emptyset$ and $\langle\rangle$ otherwise.

The f-representation $\mathcal{T}(\mathbf{R})$ represents the relation $\mathbf{R}$ if: (i) the attributes at the same node of $\mathcal{T}$ always have equal values; and (ii) each time the recursive definition of $E(\mathbf{R}, \mathcal{X}, t)$ encounters branching in $\mathcal{T}$, the respective partition of the relation can be written as a product of relations represented by the individual branches. We next formalise this intuition.

Definition 5.9. An f-tree $\mathcal{T}$ is *valid* for a relation $\mathbf{R}$ over the same schema if:

—for each node $\mathcal{A}$ of $\mathcal{T}$, the attributes of $\mathcal{A}$ have equal values in all tuples of $\mathbf{R}$; and
—for each forest $\mathcal{U} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ of $\mathcal{T}$ and each $t \in \pi_{\text{anc}(\mathcal{U})}(\mathbf{R})$, the relation $\pi_\mathcal{U}(\sigma_{\text{anc}(\mathcal{U})=t}(\mathbf{R}))$ is a product of projections $\pi_{\mathcal{T}_i}(\sigma_{\text{anc}(\mathcal{U})=t}(\mathbf{R}))$ to $\mathcal{T}_i$.

PROPOSITION 5.10. *A relation $\mathbf{R}$ has an f-representation over an f-tree $\mathcal{T}$ iff $\mathcal{T}$ is valid for $\mathbf{R}$. Any f-representation of $\mathbf{R}$ over $\mathcal{T}$ is equal to $\mathcal{T}(\mathbf{R})$ up to commutativity of product and union.*

Example 5.11. For the relation $\mathbf{R} = \{(a, b, c) : a < b < c\}$ over $\{1, \ldots, 5\}$, and the f-tree $\mathcal{T}$ with root $\mathcal{B}$ and children $\mathcal{A}$ and $\mathcal{C}$, as given in Example 5.3, we have

$$E(\mathbf{R}, \mathcal{A}, \langle B{:}2\rangle) = \langle A{:}1\rangle, \qquad E(\mathbf{R}, \mathcal{C}, \langle B{:}2\rangle) = \langle C{:}3\rangle \cup \langle C{:}4\rangle \cup \langle C{:}5\rangle, \qquad \text{and}$$
$$E(\mathbf{R}, \{\mathcal{A}, \mathcal{C}\}, \langle B{:}2\rangle) = \langle A{:}1\rangle \times (\langle C{:}3\rangle \cup \langle C{:}4\rangle \cup \langle C{:}5\rangle),$$

and similarly for $E(\mathbf{R}, \{\mathcal{A}, \mathcal{C}\}, \langle B{:}3\rangle)$ and $E(\mathbf{R}, \{\mathcal{A}, \mathcal{C}\}, \langle B{:}4\rangle)$. The expression $E(\mathbf{R}, \mathcal{T}, \langle\rangle)$ is the union $\bigcup_{b=2}^{4} E(\mathbf{R}, \{\mathcal{A}, \mathcal{C}\}, \langle B{:}b\rangle)$. Moreover, the relation $\pi_{\{\mathcal{A},\mathcal{C}\}}(\sigma_{\mathcal{B}=2}\mathbf{R})$ is a product of projections $\langle A{:}1\rangle$ and $(\langle C{:}3\rangle \cup \langle C{:}4\rangle \cup \langle C{:}5\rangle)$ to $\mathcal{A}$ and $\mathcal{C}$, respectively, and similarly for $\mathcal{B} = 3$ and $\mathcal{B} = 4$. Therefore $\mathcal{T}$ is valid for $\mathbf{R}$, and $\mathcal{T}(\mathbf{R}) = E(\mathbf{R}, \mathcal{T}, \langle\rangle)$ is the unique f-representation of $\mathbf{R}$ over $\mathcal{T}$, fully shown in Example 5.3.

For $\mathbf{R}' = \{\langle 1, 1, 1\rangle, \langle 2, 1, 2\rangle\}$ from Example 5.5 and the same f-tree $\mathcal{T}$,

$$E(\mathbf{R}', \mathcal{A}, \langle B{:}1\rangle) = \langle A{:}1\rangle \cup \langle A{:}2\rangle, \qquad E(\mathbf{R}', \mathcal{C}, \langle B{:}2\rangle) = \langle C{:}1\rangle \cup \langle C{:}2\rangle, \qquad \text{and}$$
$$E(\mathbf{R}', \{\mathcal{A}, \mathcal{C}\}, \langle B{:}1\rangle) = (\langle A{:}1\rangle \cup \langle A{:}2\rangle) \times (\langle C{:}1\rangle \cup \langle C{:}2\rangle),$$

with $E'(\mathbf{R}, \mathcal{T}, \langle\rangle) = E'(\mathbf{R}, \{\mathcal{A}, \mathcal{C}\}, \langle B{:}1\rangle)$. However, the relation $\pi_{\{\mathcal{A},\mathcal{C}\}}(\sigma_{\mathcal{B}=1}\mathbf{R}') = \langle A{:}1\rangle \times \langle C{:}1\rangle \cup \langle A{:}2\rangle \times \langle C{:}2\rangle$ is not the product of projections to A and C, so $\mathcal{T}$ is not valid for $\mathbf{R}'$ and $[\![\mathcal{T}(\mathbf{R}')]\!] \neq \mathbf{R}'$.

5.3. D-Trees for D-Representations

D-trees are f-trees where we make explicit the dependencies of attributes. This (in)dependency information can be effectively used to detect repetitions of expressions in f-representations, and to construct d-representations where the repetitions are avoided, hence increasing the succinctness of the representation.

Definition 5.12. A d-tree $\mathcal{T}^{\uparrow}$ is an f-tree $\mathcal{T}$ in which each node $\mathcal{A}$ is annotated by a set of attributes $\text{key}(\mathcal{A})$ such that:

—$\text{key}(\mathcal{A}) \subseteq \text{anc}(\mathcal{A})$;
—$\text{key}(\mathcal{A})$ is a union of nodes; and
—for any child $\mathcal{B}$ of $\mathcal{A}$, $\text{key}(\mathcal{B}) \subseteq \text{key}(\mathcal{A}) \cup \mathcal{A}$.

The set $\text{key}(\mathcal{A})$ specifies those ancestor attributes of $\mathcal{A}$ on which the attributes in the subtree rooted at $\mathcal{A}$ may depend. Naturally, if $\mathcal{B}$ is a child of $\mathcal{A}$ and $\mathcal{T}_A$ may only depend on $\text{key}(\mathcal{A})$, then $\mathcal{T}_{\mathcal{B}}$ may only depend on $\text{key}(\mathcal{A}) \cup \mathcal{A}$; this is stipulated by the last condition in Definition 5.12. For an example d-tree, see, for instance, Figure 4, right.

In the sequel, we always denote by $\mathcal{T}^{\uparrow}$ a d-tree and by $\mathcal{T}$ its underlying f-tree. All notation for f-trees carries over to d-trees. We also define $\text{key}(\mathcal{T}_{\mathcal{A}}) = \text{key}(\mathcal{A})$ for any subtree $\mathcal{T}_{\mathcal{A}}$, and $\text{key}(\mathcal{U}) = \bigcup_i \text{key}(\mathcal{T}_i)$ for any forest $\mathcal{U}$ of subtrees $\{\mathcal{T}_i\}$.

5.4. Constructive Definition of D-Representations over D-Trees

We now explicitly define the d-representation $\mathcal{T}^{\uparrow}(\mathbf{R})$ for a d-tree $\mathcal{T}^{\uparrow}$ and relation $\mathbf{R}$. The d-representation is defined similarly to the f-representation $\mathcal{T}(\mathbf{R})$, except now the expressions $D(\mathbf{R}, \mathcal{X}, t)$ only depend on $t \in \text{key}(\mathcal{X})$ instead of $t \in \text{anc}(\mathcal{X})$. We also establish conditions on the relation $\mathbf{R}$ under which the d-representation over $\mathcal{T}^{\uparrow}$ exists, and show that f-representations over f-trees are a special case of d-representations over d-trees.

Definition 5.13. Let $\mathcal{T}^{\uparrow}$ be a d-tree and $\mathbf{R}$ a relation over the same schema. We define $\mathcal{T}^{\uparrow}(\mathbf{R})$ to be the set of expressions $D(\mathbf{R}, \mathcal{X}, t)$ for all subtrees or forests $\mathcal{X}$ in $\mathcal{T}$ and all $t \in \pi_{\text{key}(\mathcal{X})}(\mathbf{R})$, where the expressions $D(\mathbf{R}, \mathcal{X}, t)$ are defined as follows.

—For any leaf $\mathcal{A}$,

$$D(\mathbf{R}, \mathcal{A}, t) = \textstyle\bigcup_{a \in A} \langle \mathcal{A}{:}a\rangle,$$

where $A = \pi_{A_1}\sigma_{\text{key}(\mathcal{A})=t}\mathbf{R}$ and $A_1 \in \mathcal{A}$.
—For any subtree $\mathcal{T}_{\mathcal{A}}$ with root $\mathcal{A}$ and a forest of children $\mathcal{U} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$,

$$D(\mathbf{R}, \mathcal{T}_{\mathcal{A}}, t) = \textstyle\bigcup_{a \in A} \langle \mathcal{A}{:}a\rangle \times {}^{\uparrow}D(\mathbf{R}, \mathcal{U}, \pi_{\text{key}(\mathcal{U})}(t \times \langle \mathcal{A}{:}a\rangle)),$$

where $A = \pi_{A_1}\sigma_{\text{key}(\mathcal{A})=t}\mathbf{R}$ and $A_1 \in \mathcal{A}$.

—For any nonempty forest $\{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$,

$$D(\mathbf{R}, \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}, t) = {}^{\uparrow}D(\mathbf{R}, \mathcal{T}_1, \pi_{\mathrm{key}(\mathcal{T}_1)}t) \times \cdots \times {}^{\uparrow}D(\mathbf{R}, \mathcal{T}_k, \pi_{\mathrm{key}(\mathcal{T}_k)}t).$$

If $\mathcal{T}$ is empty, then $\mathcal{T}^{\uparrow}(\mathbf{R})$ is the set consisting of the expression $D(\mathbf{R}, \mathcal{T}, \langle\rangle)$ that is $\emptyset$ if $\mathbf{R} = \emptyset$ and $\langle\rangle$ otherwise.

The definition of $\mathcal{T}^{\uparrow}(\mathbf{R})$ is the same as $\mathcal{T}(\mathbf{R})$ except that its expressions are not inlined recursively but only referenced in other expressions, and all expressions $E(\mathbf{R}, \mathcal{X}, t)$ whose context t agrees on the values of $\mathrm{key}(\mathcal{X})$ have been replaced by a single expression $D(\mathbf{R}, \mathcal{X}, \pi_{\mathrm{key}(\mathcal{X})}t)$. If all replaced expressions $E(\mathbf{R}, \mathcal{X}, t)$ were indeed equal to $D(\mathbf{R}, \mathcal{X}, \pi_{\mathrm{key}(\mathcal{X})}t)$, then the traversal of $\mathcal{T}^{\uparrow}(\mathbf{R})$ is $\mathcal{T}(\mathbf{R})$ and they represent the same relation. We next specify a precise condition on $\mathbf{R}$ and $\mathcal{T}^{\uparrow}$ when this happens.

Definition 5.14. A d-tree $\mathcal{T}^{\uparrow}$ is *valid* for a relation $\mathbf{R}$ if:

—$\mathcal{T}$ is valid for $\mathbf{R}$ and
—for any node $\mathcal{A}$ and any tuples t_1, t_2 over $\mathrm{anc}(\mathcal{A})$ such that $\pi_{\mathrm{key}(\mathcal{A})}(t_1) = \pi_{\mathrm{key}(\mathcal{A})}(t_2)$, it holds that $\pi_{\mathcal{T}_{\mathcal{A}}}(\sigma_{\mathrm{anc}(\mathcal{A})=t_1}(\mathbf{R})) = \pi_{\mathcal{T}_{\mathcal{A}}}(\sigma_{\mathrm{anc}(\mathcal{A})=t_2}(\mathbf{R}))$.

The first condition ensures that $\mathcal{T}(\mathbf{R})$ represents $\mathbf{R}$ and the second condition that the traversal of $\mathcal{T}^{\uparrow}(\mathbf{R})$ is $\mathcal{T}(\mathbf{R})$. This is formalised in the following proposition.

PROPOSITION 5.15. *If $\mathcal{T}^{\uparrow}$ is valid for $\mathbf{R}$, then $\mathcal{T}^{\uparrow}(\mathbf{R})$ is a d-representation of $\mathbf{R}$ and its traversal is $\mathcal{T}(\mathbf{R})$.*

A d-representation is normal (deterministic) if its traversal is normal (respectively, deterministic). Since any f-representation $\mathcal{T}(\mathbf{R})$ is normal and deterministic, by Proposition 5.15 it follows that any d-representation $\mathcal{T}^{\uparrow}(\mathbf{R})$ is normal and deterministic.

If $\mathcal{T}$ is a valid f-tree for a relation $\mathbf{R}$, the annotation $\mathrm{key}(\mathcal{A}) = \mathrm{anc}(\mathcal{A})$ for all nodes $\mathcal{A}$ yields a valid d-tree $\mathcal{T}^{\uparrow}$, the subexpressions $D(\mathbf{R}, \mathcal{X}, t)$ of $\mathcal{T}^{\uparrow}(\mathbf{R})$ correspond one-to-one to the subexpressions $E(\mathbf{R}, \mathcal{X}, t)$ of $\mathcal{T}(\mathbf{R})$, and each $D(\mathbf{R}, \mathcal{X}, t)$ is used at most once, so there is no sharing of subexpressions. F-representations over f-trees are thus a special case of d-representations over d-trees.

Example 5.16. Consider the relation $\mathbf{R} = \{(a, b, c) : 1 \leq a < b < c \leq 5\}$ from Example 5.3 and the f-tree $\mathcal{T}_2$ with root $\mathcal{A}$, child $\mathcal{B}$, and its child $\mathcal{C}$. The corresponding f-representation is (singleton types omitted)

$$\begin{aligned}\mathcal{T}_2(\mathbf{R}) \;=\; & \langle 1\rangle \times \big(\langle 2\rangle \times (\langle 3\rangle \cup \langle 4\rangle \cup \langle 5\rangle) \;\cup\; \langle 3\rangle \times (\langle 4\rangle \cup \langle 5\rangle) \;\cup\; \langle 4\rangle \times \langle 5\rangle\big) \\ & \cup \langle 2\rangle \times \big(\langle 3\rangle \times (\langle 4\rangle \cup \langle 5\rangle) \;\cup\; \langle 4\rangle \times \langle 5\rangle\big) \\ & \cup \langle 3\rangle \times \langle 4\rangle \times \langle 5\rangle,\end{aligned}$$

where the expressions $E(\mathbf{R}, \mathcal{C}, \langle 1, 3\rangle) = E(\mathbf{R}, \mathcal{C}, \langle 2, 3\rangle) = (\langle 4\rangle \cup \langle 5\rangle)$ and $E(\mathbf{R}, \mathcal{C}, \langle 1, 4\rangle) = E(\mathbf{R}, \mathcal{C}, \langle 2, 4\rangle) = E(\mathbf{R}, \mathcal{C}, \langle 3, 4\rangle) = \langle 5\rangle$ are equal because the possible values of C are determined by the value of B, independent of A.

We can set $\text{key}(\mathcal{A}) = \emptyset$, $\text{key}(\mathcal{B}) = \mathcal{A}$ and $\text{key}(\mathcal{C}) = \mathcal{B}$ to obtain the d-tree $\mathcal{T}_2^{\uparrow}$, for which

$$
\begin{aligned}
D(\mathbf{R}, \mathcal{C}, \langle B{:}2\rangle) &:= \langle C{:}3\rangle \cup \langle C{:}4\rangle \cup \langle C{:}5\rangle,\\
D(\mathbf{R}, \mathcal{C}, \langle B{:}3\rangle) &:= \langle C{:}4\rangle \cup \langle C{:}5\rangle,\\
D(\mathbf{R}, \mathcal{C}, \langle B{:}4\rangle) &:= \langle C{:}5\rangle,\\
D(\mathbf{R}, \mathcal{T}_{2\mathcal{B}}, \langle A{:}1\rangle) &:= \langle B{:}2\rangle \times {}^{\uparrow}D(\mathbf{R}, \mathcal{C}, \langle B{:}2\rangle \cup \langle B{:}3\rangle \times {}^{\uparrow}D(\mathbf{R}, \mathcal{C}, \langle B{:}3\rangle \cup \langle B{:}4\rangle \times {}^{\uparrow}D(\mathbf{R}, \mathcal{C}, \langle B{:}4\rangle),\\
D(\mathbf{R}, \mathcal{T}_{2\mathcal{B}}, \langle A{:}2\rangle) &:= \langle B{:}3\rangle \times {}^{\uparrow}D(\mathbf{R}, \mathcal{C}, \langle B{:}3\rangle \cup \langle B{:}4\rangle \times {}^{\uparrow}D(\mathbf{R}, \mathcal{C}, \langle B{:}4\rangle),\\
D(\mathbf{R}, \mathcal{T}_{2\mathcal{B}}, \langle A{:}2\rangle) &:= \langle B{:}4\rangle \times {}^{\uparrow}D(\mathbf{R}, \mathcal{C}, \langle B{:}4\rangle), \qquad \text{and}\\
D(\mathbf{R}, \mathcal{T}_2, \langle\rangle) &:= \langle A{:}1\rangle \times {}^{\uparrow}D(\mathbf{R}, \mathcal{T}_{2\mathcal{B}}, \langle A{:}1\rangle \cup \langle A{:}2\rangle \times {}^{\uparrow}D(\mathbf{R}, \mathcal{T}_{2\mathcal{B}}, \langle A{:}2\rangle\\
&\qquad \cup \langle A{:}3\rangle \times {}^{\uparrow}D(\mathbf{R}, \mathcal{T}_{2\mathcal{B}}, \langle A{:}3\rangle).
\end{aligned}
$$

Then $\mathcal{T}_2^{\uparrow}(\mathbf{R})$ consists of the preceding expressions with root $D(\mathbf{R}, \mathcal{T}_2, \langle\rangle)$. Note, for example, that the expression $(\langle C{:}4\rangle \cup \langle C{:}5\rangle)$ with two singletons is repeated twice in $\mathcal{T}_2(\mathbf{R})$, but in $\mathcal{T}_2^{\uparrow}(\mathbf{R})$ it is defined only once as $D(\mathbf{R}, \mathcal{C}, \langle B{:}3\rangle)$ and then referred to symbolically.

5.5. Computing F-Representations and D-Representations

We show that, for any relation $\mathbf{R}$ and a d-tree $\mathcal{T}^{\uparrow}$ valid for $\mathbf{R}$, the constructive Definition 5.13 of the d-representation $\mathcal{T}^{\uparrow}(\mathbf{R})$ can be converted into an algorithm computing $\mathcal{T}^{\uparrow}(\mathbf{R})$ with quasilinear data complexity (Proposition 5.17). The result naturally subsumes the case of f-representations over f-trees and is asymptotically optimal up to the logarithmic factor (with respect to data complexity).

PROPOSITION 5.17. *Given a relation $\mathbf{R}$ over schema $\mathcal{S}$ and a d-tree $\mathcal{T}^{\uparrow}$ valid for $\mathbf{R}$, the d-representation $\mathcal{T}^{\uparrow}(\mathbf{R})$ can be computed in time $O(|\mathbf{R}| \cdot \log |\mathbf{R}| \cdot |\mathcal{S}| \cdot |\mathcal{T}|^2)$.*

PROOF. By Definition 5.13, the d-representation $\mathcal{T}^{\uparrow}(\mathbf{R})$ consists of the following expressions. For each subtree $\mathcal{T}_{\mathcal{A}}$ and any $t \in \pi_{\text{key}(\mathcal{A})}(\mathbf{R})$, we have

$$D(\mathbf{R}, \mathcal{T}_{\mathcal{A}}, t) = \textstyle\bigcup_{a \in A} \langle \mathcal{A}{:}a\rangle \times {}^{\uparrow}D(\mathbf{R}, \mathcal{U}, \pi_{\text{key}(\mathcal{U})}(t \times \langle \mathcal{A}{:}a\rangle)),$$

where $A = \pi_{\mathcal{A}}\sigma_{\text{key}(\mathcal{A})=t}\mathbf{R}$, and $\mathcal{U}$ is the forest of children of $\mathcal{A}$ (if $\mathcal{U}$ is empty, the expression $D(\mathbf{R}, \mathcal{U}, \cdot)$ is omitted). For each forest $\mathcal{U}$ of sibling subtrees $\mathcal{T}_1, \ldots, \mathcal{T}_k$ and any $t \in \pi_{\text{key}(\mathcal{U})}(\mathbf{R})$, we have

$$D(\mathbf{R}, \mathcal{U}, t) = {}^{\uparrow}D(\mathbf{R}, \mathcal{T}_1, \pi_{\text{key}(\mathcal{T}_1)}t) \times \cdots \times {}^{\uparrow}D(\mathbf{R}, \mathcal{T}_k, \pi_{\text{key}(\mathcal{T}_k)}t).$$

Algorithm. For any node $\mathcal{A}$, sort the entire relation $\mathbf{R}$ by $\text{key}(\mathcal{A}) \cup \mathcal{A}$, so that it is grouped by $\text{key}(\mathcal{A})$, and those groups corresponding to $t \in \pi_{\text{key}(\mathcal{A})}\mathbf{R}$ are grouped by $\mathcal{A}$, the subgroups corresponding to $\pi_{\mathcal{A}}\sigma_{\text{key}(\mathcal{A})=t}\mathbf{R}$. All expressions $D(\mathbf{R}, \mathcal{T}_{\mathcal{A}}, t)$ can be constructed from this information in one linear pass through $\mathbf{R}$. Similarly, for any forest $\mathcal{U}$, group the entire relation by $\text{key}(\mathcal{U})$ and, in one pass through $\mathbf{R}$, construct the expression $D(\mathbf{R}, \mathcal{U}, t)$ for each $t \in \pi_{\text{key}(\mathcal{U})}(\mathbf{R})$. In the implementation, index the expressions $D(\mathbf{R}, \mathcal{X}, t)$ only by $\mathcal{X}$ and t, as $\mathbf{R}$ is the same for all expressions. To create a parse graph of the d-representation, construct a parse graph of each expression separately, insert all expression names into an associative map, and redirect all edges ending at a reference ${}^{\uparrow}D$ to the root of D.

Running time. The relation has $|\mathbf{R}|$ tuples of size $|\mathcal{S}|$ each. Sorting the relation thus takes time $O(|\mathbf{R}| \cdot \log |\mathbf{R}| \cdot |\mathcal{S}|)$. Constructing each reference ${}^{\uparrow}D(\mathbf{R}, \mathcal{X}, t)$ takes time $O(|\mathcal{S}|)$ and there are $O(|\mathcal{T}|)$ references in each expression. There are at most $|\mathbf{R}|$ expressions computed in each linear pass. There is one linear pass for each node and forest in $\mathcal{T}$, so the total running time of the algorithm is $O(|\mathbf{R}| \cdot \log |\mathbf{R}| \cdot |\mathcal{S}| \cdot |\mathcal{T}|^2)$. Each operation in an associative map of $O(|\mathbf{R}| \cdot |\mathcal{T}|^2)$ names takes time $O(\log |\mathbf{R}| + \log |\mathcal{T}|)$, so creating the parse tree does not increase the time complexity. □

6. FACTORISABILITY OF QUERY RESULTS

In this section we study the factorisability of results of conjunctive queries using f-representations and d-representations over f-trees and, respectively, d-trees. In particular, for any conjunctive query Q, we characterise the f-trees (d-trees) over which all results of Q admit an f-representation (and, respectively, a d-representation). This allows to choose an f-tree or a d-tree for representing a query result by static analysis of the query syntax only, without consulting the input data. In conjunction with the upper bounds on representations over f-trees and d-trees given in Section 7, the results of this section yield asymptotic upper bounds on the resulting factorisation size.

We consider without loss of generality f-trees and d-trees whose nodes correspond bijectively to the equivalence classes of head attributes in the input query Q; a detailed justification is given in Remark 6.3.

6.1. F-Trees for Queries

In Section 3 we defined the notion of Q-dependent attributes for a conjunctive query Q, and showed that Q-dependent attributes can be dependent in some results of Q. We use this property to characterise those f-trees that factorise all results of Q.

PROPOSITION 6.1. *Let Q be a conjunctive query and $\mathcal{T}$ an f-tree whose nodes are equivalence classes of attributes of Q. Then, $Q(\mathbf{D})$ has an f-representation over $\mathcal{T}$ for any database $\mathbf{D}$ iff any two Q-dependent nodes lie along a root-to-leaf path in $\mathcal{T}$.*

The intuition behind Proposition 6.1 is as follows. Setting aside the condition that attributes in a node have equal values, an f-tree $\mathcal{T}$ is valid for a relation $\mathbf{R}$ if any two sibling subtrees in $\mathcal{T}$ are independent conditioned on their common ancestors in $\mathcal{T}$. Since Q-dependent attributes are dependent in some results of Q, they cannot lie in sibling subtrees of $\mathcal{T}$, that is, they must lie on a root-to-leaf path.

PROOF OF PROPOSITION 6.1. We first show that the path condition is necessary. Let $\mathcal{A}$ and $\mathcal{B}$ be nodes that do not lie along a root-to-leaf path in $\mathcal{T}$; they lie in sibling subtrees $\mathcal{T}_a$ and $\mathcal{T}_b$ of some forest $\mathcal{U}$ of $\mathcal{T}$. If $Q(\mathbf{D})$ has an f-representation over $\mathcal{T}$ then, in this f-representation, for any c over $\mathrm{anc}(\mathcal{U})$ the fragment $\pi_{\mathcal{U}}\sigma_{\mathrm{anc}(\mathcal{U})=c}Q(\mathbf{D})$ is represented by a single expression $E(\mathbf{R},\mathcal{U},c)$, and hence $\sigma_{\mathrm{anc}(\mathcal{U})=c}Q(\mathbf{D})$ is a product of its projections $\pi_{\mathcal{U}}\sigma_{\mathrm{anc}(\mathcal{U})=c}Q(\mathbf{D})$ and $\pi_{\mathcal{T}\setminus\mathcal{U}}\sigma_{\mathrm{anc}(\mathcal{U})=c}Q(\mathbf{D})$. Moreover, for any c, the relation $\pi_{\mathcal{U}}\sigma_{\mathrm{anc}(\mathcal{U})=c}Q(\mathbf{D})$ is a product of its projections to the individual subtrees, including $\pi_{\mathcal{T}_a}Q(\mathbf{D})$ and $\pi_{\mathcal{T}_b}Q(\mathbf{D})$. Therefore,

$$Q(\mathbf{D}) = \pi_{\mathcal{T}_a\cup\mathrm{anc}(\mathcal{U})}Q(\mathbf{D}) \bowtie_{\mathrm{anc}(\mathcal{U})} \pi_{\mathcal{T}_b\cup\mathrm{anc}(\mathcal{U})}Q(\mathbf{D}) \bowtie_{\mathrm{anc}(\mathcal{U})} \pi_{\mathcal{T}\setminus\mathcal{T}_a\setminus\mathcal{T}_b}Q(\mathbf{D}),$$

so $\mathcal{A}$ and $\mathcal{B}$ are independent conditioned on $\mathrm{anc}(\mathcal{U})$ in $Q(\mathbf{D})$ and hence not Q-dependent. It follows that any two Q-dependent nodes do lie on a root-to-leaf path in $\mathcal{T}$.

Conversely, we prove that if any two dependent nodes lie along a root-to-leaf path then, for any forest $\mathcal{U}$ of subtrees $\mathcal{T}_j$ in $\mathcal{T}$, and any tuple $t \in \pi_{\mathrm{anc}(\mathcal{U})}(Q(\mathbf{D}))$, the relation $\pi_{\mathcal{U}}(\sigma_{\mathrm{anc}(\mathcal{U})=t}(Q(\mathbf{D})))$ is a product of its projections to the subtrees $\mathcal{T}_j$. Let T_j be the set of attributes in $\mathcal{T}_j$ together with all equivalent and dependent attributes. Then any relation with attributes in some T_j has all its attributes in T_j or equivalent to $\mathrm{anc}(\mathcal{U})$, and hence $\sigma_{\mathrm{anc}(\mathcal{U})=t}(\times_i\mathbf{R}_i)$ is a product of its projections $\pi_{\mathcal{T}_j}\sigma_{\mathrm{anc}(\mathcal{U})=t}(\times_i\mathbf{R}_i)$, $\mathrm{anc}(\mathcal{U})$, and the remaining attributes. Thus

$$\pi_{\mathcal{U}}(\sigma_{\mathrm{anc}(\mathcal{U})=t}(Q(\mathbf{D}))) = \pi_{\mathcal{U}}(\sigma_{\mathrm{anc}(\mathcal{U})=t}(\pi_{\mathcal{P}}(\sigma_{\psi}(\times_i\mathbf{R}_i)))) = \pi_{\mathcal{U}}(\sigma_{\psi}(\sigma_{\mathrm{anc}(\mathcal{U})=t}(\times_i\mathbf{R}_i)))$$

is a product of its projections to $\mathcal{T}_j$. □

The condition in Proposition 6.1 is called the *path condition*. Any f-tree satisfying the path condition is *valid* for the query Q: we call it an f-tree of Q. Proposition 6.1 shows that an f-tree is valid for a query Q if and only if it is valid for all possible results of Q.

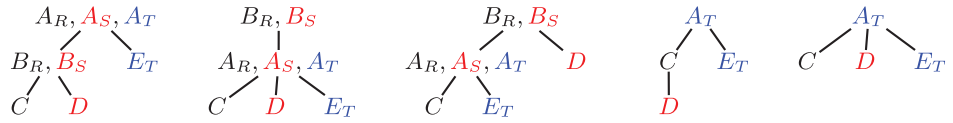


Fig. 1. Left to right: Two valid f-trees $\mathcal{T}_1$ and $\mathcal{T}_2$ and one invalid f-tree for query Q_1 in Example 6.2. A valid and an invalid f-tree for the query $\pi_{A_T,C,D,E_T} Q_1$ in Example 6.2.

In the simpler case of an equi-join query Q (thus without projection), two nodes are Q-dependent whenever they contain attributes that belong to the same relation. The path condition for an equi-join query then states that the attributes of any relation must lie along a root-to-leaf path in $\mathcal{T}$.

Example 6.2. 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 equi-join query $Q_1 = \sigma_\psi(R \times S \times T)$ with $\psi = (A_R = A_S = A_T, B_R = B_S)$. The first and second f-trees in Figure 1 are valid for Q_1. The third f-tree is invalid since the attributes A_S and D are both from relation S and hence Q_1-dependent, but are not on a common root-to-leaf path.

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

Remark 6.3. Proposition 6.1 only considers those f-trees whose node labels coincide with the attribute classes of Q. For other f-trees of the same schema as Q, the characterisation can be extended as follows. If two attributes not equivalent in Q label the same node in $\mathcal{T}$, then $Q(\mathbf{D})$ does not always have an f-representation over $\mathcal{T}$. If two attributes equivalent in Q are in different nodes of $\mathcal{T}$ and have no equivalent common ancestor, then $Q(\mathbf{D})$ also need not have an f-representation over $\mathcal{T}$. The f-trees left out are those where several nodes have attributes from the same class and, for each class, among those nodes containing attributes of this class, there is one which is an ancestor of the others. For any such f-tree, $Q(\mathbf{D})$ always has an f-representation over $\mathcal{T}$, but the f-tree constructed by pushing up all attributes of a class to the topmost node labelled by an attribute in this class defines f-representations with smaller or equal size. For the purpose of this work, Proposition 6.1 thus characterises all interesting f-trees.

6.2. D-Trees for Queries

We now extend our characterisation of f-trees, over which all results of a given query Q admit an f-representation, to d-trees and d-representations: The underlying f-tree of the d-tree must be valid for Q and, for any node $\mathcal{B}$ of $\mathcal{T}^\uparrow$, the subtree $\mathcal{T}_\mathcal{B}$ may only be dependent on those ancestors of $\mathcal{B}$ that are in the set key($\mathcal{B}$).

Definition 6.4. A d-tree $\mathcal{T}^\uparrow$ is *valid* for a query Q if the f-tree $\mathcal{T}$ is valid for Q and there is no node $\mathcal{B}$ with an ancestor $\mathcal{A} \not\subseteq$ key($\mathcal{B}$) and a descendant $\mathcal{C}$ such that $\mathcal{A}$ and $\mathcal{C}$ are Q-dependent.

PROPOSITION 6.5. *Let Q be a conjunctive query and let $\mathcal{T}^\uparrow$ be a d-tree whose nodes are equivalence classes of attributes of Q. Then $Q(\mathbf{D})$ has a d-representation over $\mathcal{T}^\uparrow$ for any database $\mathbf{D}$ iff $\mathcal{T}^\uparrow$ is valid for Q.*

Consider a query Q and a valid f-tree $\mathcal{T}$. We can obtain a valid d-tree by defining key($\mathcal{B}$) = anc($\mathcal{B}$) for all nodes $\mathcal{B}$, since then no node $\mathcal{B}$ has an ancestor $\mathcal{A} \not\subseteq$ key($\mathcal{B}$), and the validity condition holds vacuously. Furthermore, the nodes of anc($\mathcal{B}$) can be divided

into those that are Q-dependent on some node from $\mathcal{T}_\mathcal{B}$, and those that are not. Each node Q-dependent on a node in $\mathcal{T}_\mathcal{B}$ must be in key($\mathcal{B}$) for the d-tree to be valid, yet the others need not be in key($\mathcal{B}$). As we show in the following section, by shrinking the set of keys at a node, the resulting d-representation decreases in size. Therefore we will often consider the minimal possible keys of an f-tree.

Definition 6.6. The *minimal d-tree* of an f-tree $\mathcal{T}$ for a query Q is the d-tree $\mathcal{T}^\uparrow$ where, for each node $\mathcal{B}$, key($\mathcal{B}$) is the set of all nodes of anc($\mathcal{B}$) which are Q-dependent on some node from $\mathcal{T}_\mathcal{B}$.

Example 6.7. Consider the query Q_2 from Example 7.17 whose hypergraph is depicted in Figure 2, left, and the d-tree $\mathcal{T}_4^\uparrow$ depicted in Figure 2, right. The f-tree $\mathcal{T}_4$ is a single root-to-leaf path, so valid for Q_2. The only node N in $\mathcal{T}_4^\uparrow$ for which key(N) $\neq$ anc(N) is the leaf $\mathcal{E}$; no attributes from $\mathcal{B}$ are in key($\mathcal{E}$). However, no attributes from the subtree $\mathcal{T}_\mathcal{E} = \mathcal{E}$ are Q-dependent on $\mathcal{B}$. Therefore $\mathcal{T}_4^\uparrow$ is also valid for Q_2. In fact, it is the minimal d-tree of its underlying f-tree $\mathcal{T}_4$.

6.3. Extensions of F-Trees and D-Trees

We present an alternative characterisation of the d-trees of a conjunctive query Q via d-trees of its equi-join $\hat{Q}$. Given a d-tree $\hat{\mathcal{T}}^\uparrow$ of $\hat{Q}$, a first approach to obtain a d-tree $\mathcal{T}^\uparrow$ of Q is to remove those attributes from $\hat{\mathcal{T}}^\uparrow$ that are projected away in Q. A problem arises when all attributes of a node are projected away: the node remains without attributes and the d-tree is no longer well formed. Moreover, the expressions of the corresponding union would not be labelled by distinct singletons, and the resulting d-representation may encode duplicate products of singletons and hence cease to be deterministic. Hence we would lose the desirable property of optimal delay tuple enumeration (cf. Theorem 4.11).

Removing an empty node from the d-tree is also not always feasible as illustrated by Example 6.2: the attributes in its children subtrees may become dependent, which invalidates the d-tree. However, removing an empty leaf never invalidates the d-tree. This observation leads to an alternative characterisation of d-trees of arbitrary conjunctive queries.

Definition 6.8. An *extension* of a d-tree $\mathcal{T}^\uparrow$ (f-tree $\mathcal{T}$) of a conjunctive query Q is a d-tree $\hat{\mathcal{T}}^\uparrow$ (f-tree $\hat{\mathcal{T}}$) of the equi-join $\hat{Q}$ of Q such that $\mathcal{T}^\uparrow$ ($\mathcal{T}$) can be obtained from $\hat{\mathcal{T}}^\uparrow$ ($\hat{\mathcal{T}}$) by erasing the non-head attributes in Q and repeatedly removing empty leaf nodes.

PROPOSITION 6.9. *Let Q be a conjunctive query. A d-tree $\mathcal{T}^\uparrow$ and an f-tree $\mathcal{T}$ are valid for Q iff there exists an extension $\hat{\mathcal{T}}^\uparrow$ of $\mathcal{T}^\uparrow$ and, respectively, an extension $\hat{\mathcal{T}}$ of $\mathcal{T}$.*

PROOF. First we prove the claim for f-trees. Let $\mathcal{T}$ be an f-tree valid for Q, so that any two Q-dependent nodes lie on a single root-to-leaf path. If we add equivalent non-head attributes to the existing nodes, their Q-dependence does not change. The relations of Q can be partitioned into equivalence classes of relations that are either joined by attributes that are all projected out or that are connected by a chain of thus joined relations. For any such class $\mathcal{R}$ of relations, any two of their attributes are Q-dependent, and hence all of them lie on a single root-to-leaf path (no two of the attributes can lie in sibling subtrees). Let L be a lowest node on that path with an attribute from $\mathcal{R}$. Add a path of nodes under L, each node labelled by an equivalence class of non-head attributes from $\mathcal{R}$. Then, for any relation $R \in \mathcal{R}$, the attributes of R lie on a single root-to-leaf path extending the path ending at L. If we do this for all such classes $\mathcal{R}$ of relations, the path constraint will be satisfied for all relations of Q. Moreover, the obtained tree $\hat{\mathcal{T}}$ will be an extension of $\mathcal{T}$.

Conversely, suppose there exists an f-tree $\hat{\mathcal{T}}$ valid for $\hat{Q}$ which is an extension of $\mathcal{T}$. We will show by contradiction that $\mathcal{T}$ is valid for Q: suppose $\mathcal{T}$ is invalid, that is, that two Q-dependent nodes $\mathcal{A}$ and $\mathcal{B}$ are in sibling subtrees $\mathcal{T}_A$ and $\mathcal{T}_B$. Since $\mathcal{A}$ and $\mathcal{B}$ are Q-dependent, their respective nodes in $\hat{\mathcal{T}}$ (call them $\mathcal{C}_0$ and $\mathcal{C}_k$) contain dependent attributes. Therefore, there exist relations $R_1, \ldots, R_k$ such that R_i and R_{i+1} are joined on attributes from a node $\mathcal{C}_i$ of $\hat{\mathcal{T}}$ labelled only by non-head attributes of Q, and R_1 has an attribute in $\mathcal{C}_0$ and R_k an attribute in $\mathcal{C}_k$. Since $\hat{\mathcal{T}}$ is valid for $\hat{Q}$, the attributes of each R_i lie on a single root-to-leaf path. However, since $\mathcal{C}_0 \in \mathcal{T}_A$ and $\mathcal{C}_k \in \mathcal{T}_B$, there must exist a relation R_i for which $\mathcal{C}_i \in \mathcal{T}_A$ and $\mathcal{C}_{i+1} \in \mathcal{T}_B$. This is a contradiction to all attributes of R_i lying on a single root-to-leaf path in $\mathcal{T}$.

Finally we prove the claim for d-trees. If a d-tree $\mathcal{T}^\uparrow$ is valid, then its f-tree $\mathcal{T}$ is valid and it has an extension $\hat{\mathcal{T}}$ by the preceding. By defining key($\mathcal{A}$) for each new node $\mathcal{A}$ to be the set of those ancestors of $\mathcal{A}$ that are dependent on $\mathcal{A}$, we create a valid d-tree $\hat{\mathcal{T}}^\uparrow$. Conversely, if there exists an extension $\hat{\mathcal{T}}^\uparrow$ of $\mathcal{T}^\uparrow$ then $\hat{\mathcal{T}}$ is an extension of $\mathcal{T}$ and hence $\mathcal{T}$ is valid for Q, and the keys key($\mathcal{A}$) on $\mathcal{T}$ make it a valid d-tree for Q since the same keys key($\mathcal{A}$) make the extension $\hat{\mathcal{T}}^\uparrow$ valid for $\hat{Q}$. □

Example 6.10. The fourth f-tree from the left in Figure 1 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 6.9.

The fifth f-tree in Figure 1 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.

7. SIZE BOUNDS

The main result of this section is a characterisation of conjunctive queries based on the size of f-representations and d-representations of their results. We summarise the characterisation in the following restatement of Theorems 7.13, 7.16, 7.22, and 7.25, proved later in this section.

THEOREM 7.1. *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* $\mathbf{D}$*, $Q(\mathbf{D})$ admits an f-representation with size $O(|\mathcal{P}| \cdot |\mathbf{D}|^{s(Q)})$; and*
—*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 $\Omega((|\mathbf{D}|/|Q|)^{s(Q)})$.*

There is also a rational number $s^\uparrow(Q)$ such that:

—*for any database* $\mathbf{D}$*, $Q(\mathbf{D})$ admits a d-representation with size $O(|\mathcal{P}|^2 \cdot |\mathbf{D}|^{s^\uparrow(Q)})$; and*
—*for any d-tree $\mathcal{T}^\uparrow$ of Q, there exist arbitrarily large databases* $\mathbf{D}$ *for which the d-representation of $Q(\mathbf{D})$ over $\mathcal{T}^\uparrow$ has size $\Omega((|\mathbf{D}|/|Q|)^{s^\uparrow(Q)})$.*

In this section we only consider non-Boolean queries; results of Boolean queries can be represented by either the nullary tuple or the empty relation, both of size 1.

The corresponding upper and lower bounds from Theorem 7.1 meet with respect to data complexity. For a fixed query Q and any database $\mathbf{D}$, $Q(\mathbf{D})$ admits an f-representation with size $O(|\mathbf{D}|^{s(Q)})$ but any f-tree defines infinitely many f-representations of size $\Omega(|\mathbf{D}|^{s(Q)})$. Similarly, $Q(\mathbf{D})$ admits a d-representation of size $O(|\mathbf{D}|^{s^\uparrow(Q)})$ but any d-tree defines infinitely many d-representations of size $\Omega(|\mathbf{D}|^{s^\uparrow(Q)})$.

The lower bounds as stated before hold with respect to a fixed f-tree or a fixed d-tree. We also generalise them to the language of all f-representations over all f-trees, and

all d-representations over all d-trees. The following is a restatement of Theorems 7.23 and 7.26.

THEOREM 7.2. *For any fixed non-Boolean query Q, there exist arbitrarily large databases $\mathbf{D}$ for which any f-representation of the result $Q(\mathbf{D})$ over any f-tree has size $\Omega(|\mathbf{D}|^{s(Q)})$, and there exist arbitrarily large databases $\mathbf{D}$ for which any d-representation of the result $Q(\mathbf{D})$ over any d-tree has size $\Omega(|\mathbf{D}|^{s^\uparrow(Q)})$.*

A precise definition of the parameters $s(Q)$ and $s^\uparrow(Q)$ characterising the query Q is given later in this section, and their relationship to each other and to other known measures will be explored in the following section.

To prove Theorems 7.1 and 7.2, we first bound the size $|D|$ of a d-representation D in terms of the number of its singletons $\|D\|$. We then derive an expression for the exact number of singletons $\|\mathcal{T}^\uparrow(\mathbf{R})\|$ of a d-representation $\mathcal{T}^\uparrow(\mathbf{R})$ as a function of the relation $\mathbf{R}$ and the d-tree $\mathcal{T}^\uparrow$. We then derive upper and lower bounds on this number in case $\mathbf{R}$ is a query result $Q(\mathbf{D})$, as functions of the query Q and the size $|\mathbf{D}|$ of the input database. We adapt these results for the special case of f-representations over f-trees.

In the following, all formal statements referring to queries Q and d-trees $\mathcal{T}^\uparrow$ or f-trees $\mathcal{T}$ assume universal quantification over all conjunctive queries Q and all d-trees $\mathcal{T}^\uparrow$ of Q or f-trees $\mathcal{T}$ of Q, unless explicitly stated otherwise.

7.1. Size and Number of Singletons

The size $|D|$ of a d-representation is defined to be the number of its singletons $\|D\|$ plus the number of empty set symbols, unions, products, and references to other expressions. For d-representations over d-trees, we show that the size is not significantly larger than the number of singletons.

LEMMA 7.3. *For any relation $\mathbf{R}$ and any d-tree $\mathcal{T}^\uparrow$ valid for $\mathbf{R}$, the size of the d-representation $\mathcal{T}^\uparrow(\mathbf{R})$ satisfies the bound $|\mathcal{T}^\uparrow(\mathbf{R})| = O(\|\mathcal{T}^\uparrow(\mathbf{R})\| \cdot |\mathcal{T}|)$.*

PROOF. By Definition 5.13, the d-representation $\mathcal{T}^\uparrow(\mathbf{R})$ consists of factorised expressions of the form

$$D(\mathbf{R}, \mathcal{T}_\mathcal{A}, t) = \bigcup_{a\in A}\langle \mathcal{A}:a\rangle \times {}^\uparrow D(\mathbf{R}, \mathcal{U}, t') \quad \text{for subtrees } \mathcal{T}_\mathcal{A}, \text{ and}$$
$$D(\mathbf{R}, \mathcal{U}, t) = {}^\uparrow D(\mathcal{T}_1, t'_1) \times \cdots \times {}^\uparrow D(\mathcal{T}_k, t'_k) \quad \text{for forests } \mathcal{U},$$

where in the first expression the reference ${}^\uparrow D(\mathbf{R}, \mathcal{U}, t')$ is omitted if $\mathcal{U}$ is empty, $\langle \mathcal{A}:a\rangle$ is the product of singletons $\langle A_i:a\rangle$ for all $A_i \in \mathcal{A}$, and $\mathcal{T}_1, \ldots, \mathcal{T}_k$ are the trees comprising the forest $\mathcal{U}$. The exact definition of tuples t' and t'_i is not important. The size of each term in the union in $\mathbf{D}(\mathbf{R}, \mathcal{T}_\mathcal{A}, t)$ is $O(\mathcal{A})$. The size of each expression $D(\mathbf{R}, \mathcal{U}, t)$ is $O(k) = O(|\mathcal{T}|)$. Moreover, each expression $D(\mathbf{R}, \mathcal{U}, t)$ where $\mathcal{U}$ is a forest is referenced in at least one $D(\mathbf{R}, \mathcal{T}_\mathcal{A}, t)$ where $\mathcal{T}_\mathcal{A}$ is a subtree, apart from a possible root expression $D(\mathbf{R}, \mathcal{T}, \langle\rangle) = {}^\uparrow D(\mathbf{R}, \mathcal{T}_1, \langle\rangle) \times \cdots \times {}^\uparrow D(\mathbf{R}, \mathcal{T}_k, \langle\rangle)$ if the d-tree is itself a forest. Therefore, to each product of singletons $\langle \mathcal{A}:a\rangle$ in $\mathcal{T}^\uparrow(\mathbf{R})$ we can associate its term in the union in $D(\mathbf{R}, \mathcal{T}_\mathcal{A}, t)$, and the therein referenced expression $D(\mathbf{R}, \mathcal{U}, t)$ (if any) as shown earlier, of total size $O(|\mathcal{A}| + |\mathcal{T}|)$. Per each singleton this amounts to $O(|\mathcal{T}|)$. These expressions cover the entire d-representation up to the possible root fragment of size $O(|\mathcal{T}|)$ and those union nodes of which there are at most as many as singletons. It follows that the size of $\mathcal{T}^\uparrow(\mathbf{R})$ is $O(\|\mathcal{T}^\uparrow(\mathbf{R})\| \cdot |\mathcal{T}|)$. □

For f-representations over f-trees we can tighten the bound as follows.

LEMMA 7.4. *For any relation $\mathbf{R}$ and any f-tree $\mathcal{T}$ valid for $\mathbf{R}$, the size of the f-representation $\mathcal{T}(\mathbf{R})$ satisfies the bound $|\mathcal{T}(\mathbf{R})| = O(\|\mathcal{T}(\mathbf{R})\|)$.*

PROOF. In any f-representation $\mathcal{T}(\mathbf{R})$, each union symbol and each product symbol is followed by a singleton, so there are at most as many unions and products as singletons. Since there are no empty set symbols, the result follows. □

7.2. Counting Singletons in Representations

In this section we derive an exact expression for the number of singletons of each type in a given f-representation over an f-tree, or d-representation over a d-tree.

Consider first any f-representation of the form $\mathcal{T}(\mathbf{R})$. For any attribute A in the root of $\mathcal{T}$, $\mathcal{T}(\mathbf{R})$ contains one occurrence of the singleton $\langle A{:}a\rangle$ for each A-value a in the relation $\mathbf{R}$. For any attribute B in a child of the root, and for each A-value a, $\mathcal{T}(\mathbf{R})$ contains a singleton $\langle B{:}b\rangle$ (inside a subexpression over $\mathcal{T}_B$) for each B-value b in $\sigma_{A=a}\mathbf{R}$. Continuing top-down along $\mathcal{T}$, we deduce that, for any attribute C, each singleton $\langle C{:}c\rangle$ appears once for each combination of those values of the ancestor attributes of C with which it contributes to some tuple of $\mathbf{R}$. Similarly, in any d-representation $\mathcal{T}^{\uparrow}(\mathbf{R})$, the singleton $\langle C{:}c\rangle$ appears once for each combination of those values of the attributes in $\mathrm{key}(C)$ with which it contributes to some tuple of $\mathbf{R}$.

We next formalise the previous observation and express the exact number of singletons in the d-representation $\mathcal{T}^{\uparrow}(\mathbf{R})$ as a function of $\mathbf{R}$ and $\mathcal{T}^{\uparrow}$. Recall that for an attribute A we denote by $\mathcal{A}$ the node that contains A.

LEMMA 7.5. *Let $\mathcal{T}^{\uparrow}(\mathbf{R})$ be the d-representation of a relation $\mathbf{R}$ over a nonempty d-tree $\mathcal{T}^{\uparrow}$, A an attribute of $\mathbf{R}$, and x a value.*

—The number of occurrences of the singleton $\langle A{:}x\rangle$ in $\mathcal{T}^{\uparrow}(\mathbf{R})$ is $|\pi_{\mathrm{key}(\mathcal{A})}\sigma_{A=x}\mathbf{R}|$.
—The number of occurrences of A-singletons in $\mathcal{T}^{\uparrow}(\mathbf{R})$ is $|\pi_{\mathrm{key}(\mathcal{A})\cup\mathcal{A}}\mathbf{R}|$.
—The number of singletons in $\mathcal{T}^{\uparrow}(\mathbf{R})$ is $\|\mathcal{T}^{\uparrow}(\mathbf{R})\| = \sum_{A\in schema(\mathbf{R})} |\pi_{\mathrm{key}(\mathcal{A})\cup\mathcal{A}}\mathbf{R}|$.

PROOF. The singleton $\langle A{:}x\rangle$ occurs in expressions of the form $D(\mathbf{R}, \mathcal{T}^{\uparrow}{}_{\mathcal{A}}, t)$. In particular, by Definition 5.13 it occurs exactly once for each $t \in \pi_{\mathrm{key}(\mathcal{A})}\mathbf{R}$ such that $x \in \sigma_{\mathrm{key}(\mathcal{A})=t}\mathbf{R}$. These are exactly the $t \in \pi_{\mathrm{key}(\mathcal{A})}\sigma_{A=x}\mathbf{R}$, so the number of occurrences of $\langle A{:}x\rangle$ is $|\pi_{\mathrm{key}(\mathcal{A})}\sigma_{A=x}\mathbf{R}|$. The total number of occurrences of A-singletons is thus

$$\begin{aligned}
&\textstyle\sum_{x\in\pi_A(\mathbf{R})} |\pi_{\mathrm{key}(\mathcal{A})}\sigma_{\mathcal{A}=x}(\mathbf{R})| \\
&\quad = \textstyle\sum_{x\in\pi_A(\mathbf{R})} |\pi_{\mathrm{key}(\mathcal{A})\cup\mathcal{A}}\sigma_{\mathcal{A}=x}(\mathbf{R})| \\
&\quad = \textstyle\sum_{x\in\pi_A(\mathbf{R})} |\sigma_{\mathcal{A}=x}\pi_{\mathrm{key}(\mathcal{A})\cup\mathcal{A}}(\mathbf{R})| \\
&\quad = |\textstyle\bigcup_{x\in\pi_A(\mathbf{R})} \sigma_{\mathcal{A}=x}\pi_{\mathrm{key}(\mathcal{A})\cup\mathcal{A}}(\mathbf{R})| \\
&\quad = |\pi_{\mathrm{key}(\mathcal{A})\cup\mathcal{A}}(\mathbf{R})|.
\end{aligned}$$

Finally, if $\mathcal{T}^{\uparrow}$ is nonempty then there are no nullary singletons in $\mathcal{T}^{\uparrow}(\mathbf{R})$, so $\|\mathcal{T}^{\uparrow}(\mathbf{R})\|$ is the number of typed singletons of all types, which is $\sum_{A\in\mathrm{schema}(\mathbf{R})} |\pi_{\mathrm{key}(\mathcal{A})\cup\mathcal{A}}\mathbf{R}|$. □

An analogous result for f-representations follows by noting that $\mathcal{T}(\mathbf{R}) = \mathcal{T}^{\uparrow}(\mathbf{R})$ when $\mathrm{key}(\mathcal{A}) = \mathrm{anc}(\mathcal{A})$ for all nodes $\mathcal{A}$ of the d-tree $\mathcal{T}^{\uparrow}$, and that $\mathrm{anc}(A) \cup \mathcal{A} = \mathrm{path}(A)$.

COROLLARY 7.6 (LEMMA 7.5). *Let $\mathcal{T}(\mathbf{R})$ be the f-representation of a relation $\mathbf{R}$ over a nonempty f-tree $\mathcal{T}$, A an attribute of $\mathbf{R}$, and a be a value.*

—The number of occurrences of the singleton $\langle A{:}a\rangle$ in $\mathcal{T}(\mathbf{R})$ is $|\pi_{\mathrm{anc}(A)}\sigma_{A=a}\mathbf{R}|$.
—The number of occurrences of A-singletons in $\mathcal{T}(\mathbf{R})$ is $|\pi_{\mathrm{path}(A)}\mathbf{R}|$.
—$\|\mathcal{T}(\mathbf{R})\| = \sum_{A\in schema(\mathbf{R})} |\pi_{\mathrm{path}(A)}\mathbf{R}|$.

7.3. Upper Bounds

Lemma 7.5 gives an exact expression for the number of singletons in a d-representation $\mathcal{T}^{\uparrow}(\mathbf{R})$ in terms of the relation $\mathbf{R}$ and the d-tree $\mathcal{T}^{\uparrow}$. In case $\mathbf{R}$ is a query

result $Q(\mathbf{D})$ and $\mathcal{T}^{\uparrow}$ is valid for Q, we can quantify the number of singletons in the d-representation $\mathcal{T}^{\uparrow}(Q(\mathbf{D}))$ directly in terms of the database size $|\mathbf{D}|$.

Recall from Lemma 7.5 that, for any attribute A in $\mathcal{T}^{\uparrow}$, the number of singletons of type A in $\mathcal{T}^{\uparrow}(Q(\mathbf{D}))$ is $|\pi_{\text{key}(\mathcal{A})\cup\mathcal{A}}Q(\mathbf{D})|$. By Proposition 3.2, we can bound this number from above using the $(\text{key}(\mathcal{A})\cup\mathcal{A})$-restriction of Q and $\mathbf{D}$.

COROLLARY 7.7 (PROPOSITION 3.2 AND LEMMA 7.5). *For any database* $\mathbf{D}$*, the number of occurrences of A-singletons in the d-representation* $\mathcal{T}^{\uparrow}(Q(\mathbf{D}))$ *is at most* $|Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}(\mathbf{D}_{\text{key}(\mathcal{A})\cup\mathcal{A}})|$.

PROOF. By Lemma 7.5, the number of occurrences of A-singletons in $\mathcal{T}^{\uparrow}(Q(\mathbf{D}))$ is $|\pi_{\text{key}(\mathcal{A})\cup\mathcal{A}}(Q(\mathbf{D}))|$ and, by Proposition 3.2, this is at most $|Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}(\mathbf{D}_{\text{key}(\mathcal{A})\cup\mathcal{A}})|$. □

This is a useful upper bound because any restriction of Q, as defined in Section 3, is an equi-join, and recent results [Atserias et al. 2008] give tight asymptotic bounds on the size of results of equi-joins in terms of the input size. We next give an intuitive introduction to these asymptotic bounds.

We can estimate the size $|Q(\mathbf{D})|$ of any equi-join result as a function of $|\mathbf{D}|$ and Q. Intuitively, if we can cover all attributes of the query Q by $k \leq |Q|$ of its relations, then $|Q(\mathbf{D})|$ is at most the product of the sizes of these k relations, which is at most $|\mathbf{D}|^k$. These k relations correspond to an edge cover of size k in the hypergraph of Q. The following strengthens this idea by lifting covers to a weighted fractional version.

Definition 7.8 [*Atserias et al. 2008*]. For an equi-join 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 $\{x_{R_i}\}_{i=1}^{n}$:

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

In a fractional edge cover, to each relation R_i (or edge in the query hypergraph) we assign a weight 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. By restricting the variables x_{R_i} to the values 0 and 1, we obtain the standard nonweighted version of edge cover; in the fractional version the variables can hold any positive real number (though the optimal solution is always rational).

We showed before that, given an edge cover of the hypergraph of Q_S, the result size $|Q_S(\mathbf{D}_S)|$ is bounded by the product of the sizes of the covering corresponding relations. Atserias et al. [2008] generalise this idea and show that, given a fractional edge cover of the hypergraph of Q_S where the edge R_i has weight x_{R_i}, the result size $|Q_S(\mathbf{D}_S)|$ is bounded by the weighted product $\prod_i |\mathbf{R}_i|^{x_{R_i}}$. The following lemma is an adaptation of this result.

LEMMA 7.9. *For any equi-join query* Q *and database* $\mathbf{D}$*, we have* $|Q(\mathbf{D})| \leq M^{\rho^*(Q)} \leq |\mathbf{D}|^{\rho^*(Q)}$*, where* M *is the size of the largest relation in* $\mathbf{D}$.

PROOF. For any solution $\{x_{R_i}\}$ to the fractional edge cover linear program we have $|Q(\mathbf{D})| \leq \prod_i |\mathbf{R}_i|^{x_{R_i}}$ [Atserias et al. 2008]. By considering an optimal solution, it follows that

$$|Q(\mathbf{D})| \leq \prod_i |\mathbf{R}_i|^{x_{R_i}} \leq \prod_i M^{x_{R_i}} = M^{\sum_i x_{R_i}} = M^{\rho^*(Q)} \leq |\mathbf{D}|^{\rho^*(Q)}. \qquad \square$$

Together with Corollary 7.7, this yields the following bound.

LEMMA 7.10. *For any database* $\mathbf{D}$*, the number of occurrences of A-singletons in the d-representation* $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ *is at most* $|\mathbf{D}|^{\rho^*(Q_{\text{key}(\mathcal{A})\cup\mathcal{A}})}$.

Lemma 7.10 gives an upper bound on the number of occurrences of singletons of any given attribute. We can obtain an upper bound on the total number of occurrences of singletons in the d-representation $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ by summing these bounds over all attributes of Q. A simpler bound is obtained by estimating each of the summands by the largest one. A single bound for all possible d-trees of the query Q is obtained by considering the one with the smallest bound.

Definition 7.11. Let Q be a conjunctive query. For any d-tree $\mathcal{T}^\uparrow$ of Q, define

$$s^\uparrow(\mathcal{T}^\uparrow) = \max\{\rho^*(Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}) \mid A \in \mathcal{P}\}$$

to be the maximum possible $\rho^*(Q_{\text{key}(\mathcal{A})\cup\mathcal{A}})$ over all head attributes A of Q, and

$$s^\uparrow(Q) = \min\{s^\uparrow(\mathcal{T}^\uparrow) \mid \mathcal{T}^\uparrow \text{ is a d-tree of } Q\}$$

to be the minimum possible $s^\uparrow(\mathcal{T}^\uparrow)$ over all d-trees $\mathcal{T}^\uparrow$ of Q.

COROLLARY 7.12 (LEMMA 7.10). *—For any database* $\mathbf{D}$*, the number of singletons in* $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ *is at most* $|\mathcal{P}| \cdot |\mathbf{D}|^{s^\uparrow(\mathcal{T}^\uparrow)}$.
—For any database $\mathbf{D}$*, there exists a d-representation of* $Q(\mathbf{D})$ *with at most* $|\mathcal{P}| \cdot |\mathbf{D}|^{s^\uparrow(Q)}$ *singletons.*

Using Lemma 7.3, we can turn bounds on the number of singletons into size bounds.

THEOREM 7.13. *The size of* $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ *is* $O(|\mathcal{P}|^2 \cdot |\mathbf{D}|^{s^\uparrow(\mathcal{T}^\uparrow)})$*, and for any database* $\mathbf{D}$ *there exists a d-representation of* $Q(\mathbf{D})$ *with size* $O(|\mathcal{P}|^2 \cdot |\mathbf{D}|^{s^\uparrow(Q)})$.

PROOF. If the number of singletons in $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ is at most $|\mathcal{P}| \cdot |\mathbf{D}|^{s^\uparrow(\mathcal{T}^\uparrow)}$ then, by Lemma 7.3, $|\mathcal{T}^\uparrow(Q(\mathbf{D}))|$ is $O(|\mathcal{P}| \cdot |\mathbf{D}|^{s^\uparrow(\mathcal{T}^\uparrow)} \cdot |\mathcal{T}^\uparrow|)$, which is $O(|\mathcal{P}|^2 \cdot |\mathbf{D}|^{s^\uparrow(\mathcal{T}^\uparrow)})$ since $|\mathcal{T}^\uparrow| \leq |\mathcal{P}|$. The second claim follows. □

Analogous upper bounds can be shown for the sizes of f-representations over f-trees. The number of A-singletons in an f-representation $\mathcal{T}(\mathbf{R})$ is $|\pi_{\text{path}(\mathcal{A})}\mathbf{R}|$ by Corollary 7.6. This is at most $|Q_{\text{path}(\mathcal{A})}(\mathbf{D}_{\text{path}(\mathcal{A})})|$ by Proposition 3.2, which is at most $|\mathbf{D}|^{\rho^*(Q_{\text{path}(\mathcal{A})})}$ by Lemma 7.9. Similarly to d-trees, we maximise this value over all head attributes of $\mathcal{T}$ to obtain bounds for f-representations over $\mathcal{T}$, and then minimise over all f-trees $\mathcal{T}$ of Q to obtain bounds for f-representations of results of Q.

Definition 7.14. Let Q be a conjunctive query. For any f-tree $\mathcal{T}$ of Q, define

$$s(\mathcal{T}) = \max\{\rho^*(Q_{\text{path}(\mathcal{A})}) \mid A \in \mathcal{P}\}$$

to be the maximum possible $\rho^*(Q_{\text{path}(\mathcal{A})})$ over all head attributes A of Q, and

$$s(Q) = \min\{s(\mathcal{T}) \mid \mathcal{T} \text{ is an f-tree of } Q\}$$

to be the minimum possible $s(\mathcal{T})$ over all f-trees $\mathcal{T}$ of Q.

COROLLARY 7.15 (THEOREM 7.13). *The number of singletons in* $\mathcal{T}(Q(\mathbf{D}))$ *is at most* $|\mathcal{P}| \cdot |\mathbf{D}|^{s(\mathcal{T})}$ *and, for any database* $\mathbf{D}$*, there exists an f-representation of* $Q(\mathbf{D})$ *with at most* $|\mathcal{P}| \cdot |\mathbf{D}|^{s(Q)}$ *singletons.*

Using Lemma 7.4, we obtain the bounds on f-representation size. The difference of a factor of $|\mathcal{P}|$ compared to d-representations is due to the tighter bound on f-representation size expressed in the number of singletons.

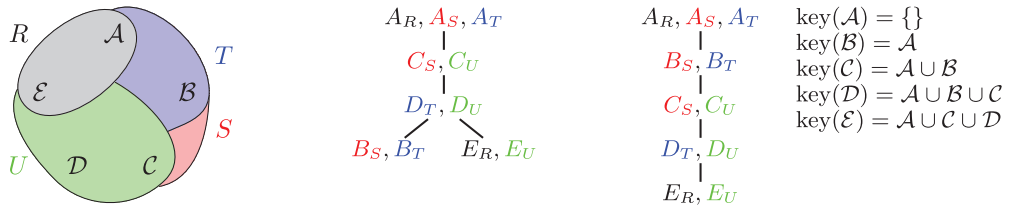


Fig. 2. Left to right: Hypergraph of query Q_2 from Example 7.17 with nodes $\mathcal{A} = \{A_R, A_S, A_T\}$, $\mathcal{B} = \{B_S, B_T\}$, $\mathcal{C} = \{C_S, C_U\}$, $\mathcal{D} = \{D_T, D_U\}$, and $\mathcal{E} = \{E_R, E_U\}$; f-trees $\mathcal{T}_3$ and $\mathcal{T}_4$ of the query Q_2; and keys of nodes of the f-tree $\mathcal{T}_4$ turning it into a d-tree $\mathcal{T}_4^{\uparrow}$.

THEOREM 7.16. *The size of $\mathcal{T}(Q(\mathbf{D}))$ is $O(|\mathcal{P}| \cdot |\mathbf{D}|^{s(\mathcal{T})})$ and, for any database $\mathbf{D}$, there exists an f-representation of $Q(\mathbf{D})$ with size $O(|\mathcal{P}| \cdot |\mathbf{D}|^{s(Q)})$.*

Example 7.17. 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 $\psi = (A_R = A_S = A_T, B_S = B_T, C_S = C_U, D_T = D_U, E_R = E_U)$. The hypergraph of Q_2 is depicted in Figure 2, left. The attribute classes of Q_2 can be covered by the two relations S and U, so $\rho^*(Q_2) \leq 2$. 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 $\rho^*(Q_2) \geq 2$. It follows that $\rho^*(Q_2) = 2$, so the result $Q_2(\mathbf{D})$ has size at most $|\mathbf{D}|^2$ for any database $\mathbf{D}$.

First consider the simpler case of f-representations over the f-trees in Figure 2. We compute $|\mathcal{T}_3(Q_2(\mathbf{D}))|$, where $\mathcal{T}_3$ is the left f-tree in Figure 2. The nodes with largest paths are $\mathcal{B} = \{B_S, B_T\}$ and $\mathcal{E} = \{E_R, E_U\}$. Consider the query restriction $Q_2^{\mathcal{B}}$. We need at least two relations to cover all attributes of $Q_2^{\mathcal{B}}$, so the edge cover number of $Q_2^{\mathcal{B}}$ 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 $\rho^*(Q_2^{\mathcal{B}}) = 3/2$. For $Q_2^{\mathcal{E}}$, the optimal solution is $x_U = 2/3$ and $x_R = x_S = x_T = 1/3$ with total cost $\rho^*(Q_2^{\mathcal{E}}) = 5/3$, and hence $s(\mathcal{T}_3) = 5/3$. It follows that the factorisation $\mathcal{T}_3(Q_2(\mathbf{D}))$ has at most $11 \cdot |\mathbf{D}|^{5/3}$ singletons, which is asymptotically smaller than the number of singletons $11 \cdot |\mathbf{D}|^2$ in the flat result.

The succinctness of representations over $\mathcal{T}_3$ is achieved by storing values of $\mathcal{B}$ and $\mathcal{E}$ independently for each combination of values of $\mathcal{A}, \mathcal{C}$, and $\mathcal{D}$, as represented by $\mathcal{B}$ and $\mathcal{E}$ lying in different branches of $\mathcal{T}_3$ under $\mathcal{A}, \mathcal{C}$, and $\mathcal{D}$. For comparison, in the right f-tree $\mathcal{T}_4$ in Figure 2, $\text{path}(\mathcal{E})$ contains all attributes of Q_2. Hence $\rho^*(Q_2^{\mathcal{E}}) = \rho^*(Q_2) = 2$, so $s(\mathcal{T}_4) = 2$ and f-representations over the f-tree $\mathcal{T}_4$ present no asymptotic saving in space compared to flat representations.

Consider now the d-tree $\mathcal{T}_4^{\uparrow}$, whose underlying f-tree is $\mathcal{T}_4$ and the node keys are as defined in Figure 2, right. Now $\text{key}(\mathcal{E}) \cup \mathcal{E} = \mathcal{A} \cup \mathcal{C} \cup \mathcal{D} \cup \mathcal{E}$ is a strict subset of $\text{path}(\mathcal{E})$, and $\rho^*(Q_{\text{key}(\mathcal{E}) \cup \mathcal{E}})$ equals $5/3$, strictly less than $\rho^*(Q_{\text{path}(\mathcal{E})}) = 2$. For all other nodes N the value $\rho^*(Q_{\text{key}(N) \cup N})$ is at most $5/3$, so d-representations over $\mathcal{T}_4^{\uparrow}$ have size at most $11 \cdot |\mathbf{D}|^{5/3}$. The succinctness of d-representations over $\mathcal{T}_4^{\uparrow}$ compared to f-representations over $\mathcal{T}_4$ is achieved by storing a union of $\mathcal{E}$-values only once for each combination of values from $\text{key}(\mathcal{E}) = \mathcal{A} \cup \mathcal{C} \cup \mathcal{D}$, and referencing this same expression for each different value of $\mathcal{B}$.

For the case of Q_2, it turns out that $\mathcal{T}_3$ is an optimal f-tree and $\mathcal{T}_4^{\uparrow}$ an optimal d-tree, so $s(Q_2) = s^{\uparrow}(Q_2) = 5/3$. It is not necessarily true that $s(Q) = s^{\uparrow}(Q)$. In Section 9 we show examples of queries with $s^{\uparrow}(Q) \ll s(Q)$.

7.4. Lower Bounds

We next show that the upper bound on the d-representation size is the best possible in the following sense. For any non-Boolean query Q and any d-tree $\mathcal{T}^{\uparrow}$ of Q, there are arbitrarily large databases for which the size of the d-representation of the query result over $\mathcal{T}^{\uparrow}$ asymptotically meets the upper bound in terms of data complexity.

By Lemma 7.5, the number of singletons of type A in $\mathcal{T}^{\uparrow}(Q(\mathbf{D}))$ is $|\pi_{\text{key}(\mathcal{A})\cup\mathcal{A}}Q(\mathbf{D})|$, and Proposition 3.2 bounds any $|\pi_S(Q(\mathbf{D}))|$ from above by $|Q_S(\mathbf{D}_S)|$, where $(Q_S, \mathbf{D}_S)$ is the S-restriction of Q and $\mathbf{D}$. The following result provides a corresponding lower bound.

LEMMA 7.18. *For any subset S of head attributes of a query Q and any database $\mathbf{D}_S$ over the schema of Q_S with largest relation of size M, there exists a database $\mathbf{D}$ with size $M \leq |\mathbf{D}| \leq |\mathbf{D}_S|$ and $|\pi_S(Q(\mathbf{D}))| \geq |Q_S(\mathbf{D}_S)|$.*

PROOF. Each relation symbol R_i^S in Q_S is the relation symbol R_i restricted to the attributes of S and attributes equivalent to those in S (denote this set S^*). Construct a database $\mathbf{D}'$ by extending each relation in $\mathbf{D}_S$ with the removed attributes: for each attribute in the schema of Q but not Q_S, we allow a single value 1, and extend each tuple in each relation by this value for these new attributes. For relations in Q but with no attributes in Q_S, the relation instance in $\mathbf{D}_S$ is $\{\langle\rangle\}$, so $\mathbf{D}'$ will consist of a single tuple with value 1 in each attribute. There is a one-to-one correspondence between the tuples of $\mathbf{D}'$ and $\mathbf{D}_S$, so $|\mathbf{D}'| = |\mathbf{D}_S|$.

Finally, we merge those relation instances in $\mathbf{D}'$ which should be equal due to self-joins. We construct the database $\mathbf{D}$ as follows. For any class $\{R_{i_1}, \ldots, R_{i_m}\}$ of relation symbols which refer to the same relation, replace the relation instances $\mathbf{R}_{i_1}, \ldots, \mathbf{R}_{i_m}$ in $\mathbf{D}'$ by a single relation instance $\mathbf{R} = \bigcup_j \mathbf{R}_{i_j}$ in $\mathbf{D}$, and interpret each of the relation symbols R_{i_j} by $\mathbf{R}$. By construction, the largest relation in $\mathbf{D}$ is at least as large as the largest relation in $\mathbf{D}_S$, so $M \leq |\mathbf{D}|$. By the union bound we have $|\mathbf{D}| \leq |\mathbf{D}'| = |\mathbf{D}_S|$, and

$$
\begin{aligned}
|\pi_S(Q(\mathbf{D}))| &= |\pi_S(\pi_{\mathcal{P}}(\sigma_{\psi}(R_1 \times \cdots \times R_n)))(\mathbf{D})| \\
&\geq |\pi_S(\pi_{\mathcal{P}}(\sigma_{\psi}(R_1 \times \cdots \times R_n)))(\mathbf{D}')| && (2)\\
&= |\pi_S(\sigma_{\psi}(R_1 \times \cdots \times R_n))(\mathbf{D}')| \\
&= |\pi_{S^*}(\sigma_{\psi}(R_1 \times \cdots \times R_n))(\mathbf{D}')| && (4)\\
&= |\pi_{S^*}(\sigma_{\psi_S}(R_1 \times \cdots \times R_n))(\mathbf{D}')| && (5)\\
&= |\sigma_{\psi_S}(\pi_{S^*}R_1 \times \cdots \times \pi_{S^*}R_n)(\mathbf{D}')| \\
&= |Q_S(\mathbf{D}_S)|.
\end{aligned}
$$

Inequality (2) holds because each relation of $\mathbf{D}'$ is a subset of the corresponding relation of $\mathbf{D}$, equality (4) holds because each attribute in S^* is equivalent to some attribute in S, and equality (5) holds because in $\mathbf{D}'$ the values in all attributes outside S^* are equal. □

In a first attempt to make the lower bound $|Q_S(\mathbf{D}_S)|$ as large as possible while keeping $|\mathbf{D}_S|$ small, we pick k attribute classes of Q_S 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 $\mathbf{D}_S$ has size $|Q_S| \cdot N$ but the result $Q_S(\mathbf{D}_S)$ has size N^k. The picked k attribute classes correspond to an independent set of k nodes in the hypergraph of Q_S.

Similar to the upper bound, we can strengthen the preceding 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 7.3. The following result forms the basis of our argument.

LEMMA 7.19 [ATSERIAS ET AL. 2008]. *For any equi-join query Q without self-joins, there exist arbitrarily large databases $\mathbf{D}$ such that $|Q(\mathbf{D})| \geq (|\mathbf{D}|/|Q|)^{\rho^*(Q)}$.*

We now use Lemmata 7.5, 7.18, and 7.19 to construct databases $\mathbf{D}$ with lower bounds on the number of A-singletons in the d-representation $\mathcal{T}^\uparrow(Q(\mathbf{D}))$.

LEMMA 7.20. *There exist arbitrarily large databases $\mathbf{D}$ such that the number of A-singletons in $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ is at least $(|\mathbf{D}|/|Q|)^{\rho^*(Q_{\text{key}(\mathcal{A})\cup\mathcal{A}})}$.*

PROOF. By Lemma 7.19 applied to $Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}$, there exist arbitrarily large databases $\mathbf{D}_{\text{key}(\mathcal{A})\cup\mathcal{A}}$ such that $|Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}(\mathbf{D}_{\text{key}(\mathcal{A})\cup\mathcal{A}})| \geq (|\mathbf{D}_{\text{key}(\mathcal{A})\cup\mathcal{A}}|/|Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}|)^{\rho^*(Q_{\text{key}(\mathcal{A})\cup\mathcal{A}})}$. By Lemma 7.18, there exists a database $\mathbf{D}$ with $|\mathbf{D}| \leq |\mathbf{D}_{\text{key}(\mathcal{A})\cup\mathcal{A}}|$ such that $|\pi_{\text{key}(\mathcal{A})\cup\mathcal{A}}(Q(\mathbf{D}))| \geq |Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}(\mathbf{D}_{\text{key}(\mathcal{A})\cup\mathcal{A}})|$. Moreover, $\mathbf{D}$ is at least as large as the largest relation of $\mathbf{D}_{\text{key}(\mathcal{A})\cup\mathcal{A}}$, so $\mathbf{D}$ also gets arbitrarily large. By Lemma 7.5, the number of A-singletons in $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ is

$$\begin{aligned}|\pi_{\text{key}(\mathcal{A})\cup\mathcal{A}}(Q(\mathbf{D}))| &\geq |Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}(\mathbf{D}_{\text{key}(\mathcal{A})\cup\mathcal{A}})| \\ &\geq (|\mathbf{D}_{\text{key}(\mathcal{A})\cup\mathcal{A}}|/|Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}|)^{\rho^*(Q_{\text{key}(\mathcal{A})\cup\mathcal{A}})} \\ &\geq (|\mathbf{D}|/|Q|)^{\rho^*(Q_{\text{key}(\mathcal{A})\cup\mathcal{A}})}.\end{aligned}$$

□

We now lift Lemma 7.20 from A-singletons to all singletons in $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ by considering that attribute A for which the lower bound $(|\mathbf{D}|/|Q|)^{\rho^*(Q_{\text{key}(\mathcal{A})\cup\mathcal{A}})}$ is the largest.

COROLLARY 7.21 (LEMMA 7.20). *There exist arbitrarily large databases $\mathbf{D}$ for which $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ has at least $(|\mathbf{D}|/|Q|)^{s^\uparrow(\mathcal{T}^\uparrow)}$ singletons.*

Since the size of a d-representation is at least the number of its singletons, we also have the following.

THEOREM 7.22. *There exist arbitrarily large databases $\mathbf{D}$ for which $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ has size $\Omega((|\mathbf{D}|/|Q|)^{s^\uparrow(\mathcal{T}^\uparrow)}) = \Omega((|\mathbf{D}|/|Q|)^{s^\uparrow(Q)})$.*

Theorem 7.22 gives a lower bound for the representation size over a given d-tree. We next give a (nontrivial) generalisation to a lower bound for the representation size in the language of d-representations over any d-tree.

THEOREM 7.23. *For a fixed query Q, there exist arbitrarily large databases $\mathbf{D}$ for which any d-representation of the result $Q(\mathbf{D})$ over any d-tree has size $\Omega(|\mathbf{D}|^{s^\uparrow(Q)})$.*

PROOF. To prove this theorem we need to strengthen the requirements on the sizes of database examples witnessing the lower bounds in Lemma 7.19 and Theorem 7.22. The changes are of a technical nature and the full proofs of the adapted versions are deferred to the electronic appendix.

LEMMA 7.19, ADAPTED. For any equi-join query Q without self-joins, there exist constants b_Q, c_Q such that, for any sufficiently large N, there exists a database $\mathbf{D}$ of size $N \leq |\mathbf{D}| \leq b_Q \cdot N$ such that $|Q(\mathbf{D})| \geq c_Q \cdot |\mathbf{D}|^{\rho^*(Q)}$.

THEOREM 7.22, ADAPTED. For any query Q there exist constants b_Q, c_Q such that, for any sufficiently large N and for any d-tree $\mathcal{T}^\uparrow$ of Q, there exists a database $\mathbf{D}_{\mathcal{T}^\uparrow}$ of size $N \leq |\mathbf{D}_{\mathcal{T}^\uparrow}| \leq b_Q \cdot N$ such that $|\mathcal{T}^\uparrow(Q(\mathbf{D}_{\mathcal{T}^\uparrow}))| \geq c_Q \cdot |\mathbf{D}_{\mathcal{T}^\uparrow}|^{s^\uparrow(Q)}$.

For any N sufficiently large, let $\mathbf{D}_{\mathcal{T}^\uparrow}$ be as in the adapted version of Theorem 7.22. Construct the database $\mathbf{D}$ as a disjoint union of $\mathbf{D}_{\mathcal{T}^\uparrow}$ for all d-trees $\mathcal{T}^\uparrow$ of Q. (Label each data element in $\mathbf{D}_{\mathcal{T}^\uparrow}$ by $\mathcal{T}^\uparrow$, so that the corresponding relations of $\mathbf{D}_{\mathcal{T}^\uparrow}$ are disjoint and, for each relation symbol of Q, construct a relation instance in $\mathbf{D}$ by taking a union of the corresponding relation instances in all $\mathbf{D}_{\mathcal{T}^\uparrow}$.) The result $Q(\mathbf{D})$ is a disjoint union

of the results $Q(\mathbf{D}_{\mathcal{T}^\uparrow})$, and for any d-tree $\mathcal{T}^\uparrow$ the d-representation $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ contains the d-representation $\mathcal{T}^\uparrow(Q(\mathbf{D}_{\mathcal{T}^\uparrow}))$, so its size is at least $c_Q \cdot |\mathbf{D}_{\mathcal{T}^\uparrow}|^{s^\uparrow(Q)}$. The size of each $\mathbf{D}_{\mathcal{T}^\uparrow}$ is at most $b_Q \cdot N$, so the size of $\mathbf{D}$ is at most $d \cdot b_Q \cdot N$, where d is the number of d-trees of Q. Therefore, for any d-tree $\mathcal{T}^\uparrow$ the d-representation $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ has size at least $b_Q \cdot (|\mathbf{D}|/(c \cdot d))^{s^\uparrow(Q)}$, which is $\Omega(|\mathbf{D}|^{s^\uparrow(Q)})$ for a fixed Q. □

For a fixed query, the upper and lower bounds on the size of d-representations of query results meet asymptotically. The fractional versions of the minimum edge 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 $s^\uparrow(Q)$ thus completely characterises queries by the representability of their results within the class of d-representations defined by d-trees.

Analogous lower bounds can be deduced for the special case of f-representations over f-trees.

LEMMA 7.24. *For any f-tree $\mathcal{T}$ of Q, there exist arbitrarily large databases $\mathbf{D}$ for which $\mathcal{T}(Q(\mathbf{D}))$ has at least $(|\mathbf{D}|/|Q|)^{s(\mathcal{T})}$ singletons.*

PROOF. Any f-tree can be seen as a d-tree with $\text{key}(\mathcal{A}) = \text{anc}(\mathcal{A})$ for all attributes $\mathcal{A}$, hence by Lemma 7.20 the number of A-singletons in any $\mathcal{T}(Q(\mathbf{D}))$ is at least $(|\mathbf{D}|/|Q|)^{\rho^*(Q_{\text{anc}(\mathcal{A})\cup\mathcal{A}})} = (|\mathbf{D}|/|Q|)^{\rho^*(Q_{\text{path}(\mathcal{A})})} \geq (|\mathbf{D}|/|Q|)^{s(\mathcal{T})}$. □

COROLLARY 7.25. *For any f-tree $\mathcal{T}$ of Q, there exist arbitrarily large databases $\mathbf{D}$ for which $\mathcal{T}(Q(\mathbf{D}))$ has size $\Omega((|\mathbf{D}|/|Q|)^{s(\mathcal{T})}) = \Omega((|\mathbf{D}|/|Q|)^{s(Q)})$.*

COROLLARY 7.26. *For a fixed query Q, there exist arbitrarily large databases $\mathbf{D}$ for which any f-representation of the result $Q(\mathbf{D})$ over any f-tree has size $\Omega(|\mathbf{D}|^{s(Q)})$.*

Example 7.27. Let us continue Example 7.17 and consider the query $Q_2 = \sigma_\psi(R(A_R, E_R) \times S(A_S, B_S, C_S) \times T(A_T, B_T, D_T) \times U(C_U, D_U, E_U))$, with $\psi = (A_R = A_S = A_T, B_S = B_T, C_S = C_U, D_T = D_U, E_R = E_U)$. Consider the left f-tree $\mathcal{T}_3$ from Figure 2. The hypergraph of $Q_2^{\mathcal{E}}$ is obtained by dropping the node $\mathcal{B}$ from the hypergraph of Q (which is depicted in Figure 2, left), and 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 $\mathcal{E}$-singletons is linear in the size of $\mathbf{D}$, yet this is much smaller than the $O(|\mathbf{D}|^{5/3})$ upper bound given by Lemma 7.24. The fractional relaxation of the maximum independent set problem allows to increase the optimal cost to 5/3, thus meeting $\rho^*(Q_2^{\mathcal{E}})$ by duality of linear programming, as follows. 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 then used in the proofs of Lemmas 7.19 and 7.20 to construct arbitrarily large databases $\mathbf{D}$ for which the number of $\mathcal{E}$-singletons in $\mathcal{T}_3(Q_2(\mathbf{D}))$ is at least $(|\mathbf{D}|/|Q_2|)^{5/3} = (|\mathbf{D}|/4)^{5/3}$.

One such database $\mathbf{D}$ would contain the relations $\mathbf{R} = [4] \times [2]$, $\mathbf{S} = [4] \times [1] \times [2]$, $\mathbf{T} = [4] \times [1] \times [2]$, and $\mathbf{U} = [2] \times [2] \times [2]$. Here $[N]$ denotes $\{1, \ldots, N\}$ and the attributes of each relation are ordered alphabetically as in the prior definition. Each relation has size 8 and the database $\mathbf{D}$ has size $32 = 8 \times |Q_2|$. The result $Q_2(\mathbf{D})$ 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 $|Q_2(\mathbf{D})| = 32 = (32/4)^{5/3} = (|\mathbf{D}|/|Q_2|)^{5/3}$. By replacing powers of 2 in this example by powers of larger integers, we can create arbitrarily large database examples with $|Q_2(\mathbf{D})| = (|\mathbf{D}|/|Q_2|)^{5/3}$.

Since all f-trees $\mathcal{T}$ for Q_2 have $s(\mathcal{T}) \geq s(Q_2) = 5/3$, the results in this section 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 $(|\mathbf{D}|/|Q_2|)^{5/3} = (|\mathbf{D}|/4)^{5/3}$.

8. CONJUNCTIVE QUERY EVALUATION

In this section we present an algorithm for computing the result of conjunctive queries directly in factorised form. In Section 5.5 we gave an algorithm that factorised a given relation over a given d-tree in quasilinear time. However, there exist queries whose results are exponentially larger than both the input database and their succinct f-representations and d-representations. The algorithms in this section compute the d-representation $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ of a query result $Q(\mathbf{D})$ directly from the input database $\mathbf{D}$, query Q, and d-tree $\mathcal{T}^\uparrow$, without an intermediate computation of the potentially large flat result $Q(\mathbf{D})$. This allows an $o(|Q(\mathbf{D})|)$ time complexity, better than for any possible algorithm computing the flat result. In particular, for an equi-join query Q, we compute the d-representation $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ with data complexity $O(|\mathbf{D}|^{s^\uparrow(\mathcal{T}^\uparrow)} \log |\mathbf{D}|)$ (Proposition 8.2), which is worst-case optimal up to the logarithmic factor. The algorithm is extended to arbitrary conjunctive queries.

Any algorithm for computing d-representations over d-trees naturally subsumes the computation of f-representations over f-trees; in particular, for any equi-join Q and its f-tree $\mathcal{T}$, we can compute $\mathcal{T}(Q(\mathbf{D}))$ with data complexity $O(|\mathbf{D}|^{s(\mathcal{T})} \log |\mathbf{D}|)$.

8.1. Computing D-Representations of Equi-Join Query Results

We show that the d-representation $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ of any equi-join query result can be computed directly from the input database $\mathbf{D}$ and the query Q, with data complexity $O(|\mathbf{D}|^{s^\uparrow(\mathcal{T}^\uparrow)} \log |\mathbf{D}|)$. The best asymptotic bound for the size of $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ is $O(|\mathbf{D}|^{s^\uparrow(\mathcal{T}^\uparrow)})$, so the algorithm is worst-case optimal up to the logarithmic factor. It is not instance optimal; for particular databases $\mathbf{D}$ the d-representation may be of size $o(|\mathbf{D}|^{s^\uparrow(\mathcal{T}^\uparrow)})$ but the algorithm may still take $\Omega(|\mathbf{D}|^{s^\uparrow(\mathcal{T}^\uparrow)})$.

The main idea of the algorithm is to evaluate individual subqueries $Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}$ for each node $\mathcal{A}$ and then stitch their results together into the d-representation of the result of Q. The results of the subqueries $Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}$ represent the largest "non-factorised" fragments of the d-representation and in fact dictate its size, as shown in Section 7. Each result $Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}(\mathbf{D})$ can be computed in traditional flat form using one of the known worst-case optimal algorithms in time $O(|\mathbf{D}|^{\rho^*(Q_{\text{key}(\mathcal{A})\cup\mathcal{A}})})$ [Ngo et al. 2012; Veldhuizen 2014].

The d-representation $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ contains one singleton $\langle\mathcal{A}{:}a\rangle$ for each tuple in $\pi_{\text{key}(\mathcal{A})\cup\mathcal{A}}(Q(\mathbf{D}))$ (more precisely, for each $t \in \pi_{\text{key}(\mathcal{A})}(Q(\mathbf{D}))$, it contains a union of $\langle\mathcal{A}{:}a\rangle$ over $a \in \pi_{\mathcal{A}}\sigma_{\text{key}(\mathcal{A})=t}(Q(\mathbf{D}))$). We first construct a larger d-representation with one singleton $\langle\mathcal{A}{:}a\rangle$ for each tuple in $Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}(\mathbf{D})$, which contains $\pi_{\text{key}(\mathcal{A})\cup\mathcal{A}}(Q(\mathbf{D}))$. Then we identify the d-representation $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ as a subset of the computed d-representation by removing all those of its subexpressions that represent the empty relation. The algorithm is given in pseudocode as Algorithm 1. We next prove its correctness and time performance.

PROPOSITION 8.1. *For any equi-join query Q, its d-tree $\mathcal{T}^\uparrow$, and database $\mathbf{D}$, Algorithm 1 computes the d-representation $\mathcal{T}^\uparrow(Q(\mathbf{D}))$.*

PROOF. First we prove that, before block 2 in Algorithm 1, the set of expressions R when interpreted as a parse graph contains the d-representation $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ as a subgraph. Since $\mathbf{R}_{\mathcal{A}} = Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}(\mathbf{D}_{\text{key}(\mathcal{A})\cup\mathcal{A}})$ contains $\pi_{\text{key}(\mathcal{A})\cup\mathcal{A}}(Q(\mathbf{D}))$ by Proposition 3.2, $\pi_{\text{key}(\mathcal{A})}(\mathbf{R}_{\mathcal{A}})$ contains $\pi_{\text{key}(\mathcal{A})}(Q(\mathbf{D}))$ and similarly $\pi_{\text{key}(\mathcal{U})}(\mathbf{R}_{\mathcal{A}})$ contains $\pi_{\text{key}(\mathcal{U})}(Q(\mathbf{D}))$ for the forest $\mathcal{U}$ under $\mathcal{A}$, so, for each expression $D(\mathbf{R}, \mathcal{X}, t)$ in $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ as per Definition 5.13, the set R also contains an expression named $D(\mathbf{R}, \mathcal{X}, t)$. Moreover, all expressions

ALGORITHM 1: Computing the d-representation of an equi-join query result.

Data: Equi-join query Q, d-tree $\mathcal{T}^\uparrow$, database $\mathbf{D}$.
Result: D-representation $\mathcal{T}^\uparrow(Q(\mathbf{D}))$.

$R \longleftarrow$ empty d-representation;

for *all nodes* $\mathcal{A}$ *in* $\mathcal{T}$ **do**
  let $\{\mathcal{T}_1, \ldots, \mathcal{T}_k\} = \mathcal{U} \longleftarrow$ forest of children subtrees of $\mathcal{A}$ in $\mathcal{T}$;
  $\mathbf{R}_\mathcal{A} \longleftarrow Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}(\mathbf{D})$ using a worst-case optimal algorithm for equi-joins;
  group $\mathbf{R}_\mathcal{A}$ by key($\mathcal{A}$);
  for $t \in \pi_{\text{key}(\mathcal{A})}(\mathbf{R}_\mathcal{A})$ **do**
    $A_t \longleftarrow \pi_\mathcal{A}\sigma_{\text{key}(\mathcal{A})=t}(\mathbf{R}_\mathcal{A})$;
    if $\mathcal{U} = \emptyset$ **then** $D(\mathcal{T}_\mathcal{A}, t) \longleftarrow \bigcup_{a\in A_t}\langle\mathcal{A}{:}a\rangle$;
      else $D(\mathcal{T}_\mathcal{A}, t) \longleftarrow \bigcup_{a\in A_t}\langle\mathcal{A}{:}a\rangle \times {}^\uparrow D(\mathcal{U}, \pi_{\text{key}(\mathcal{U})}(t \times \langle\mathcal{A}{:}a\rangle))$;
    add expression $D(\mathcal{T}_\mathcal{A}, t)$ to R;
  end
  group $\mathbf{R}_\mathcal{A}$ by key($\mathcal{U}$);
  for $t \in \pi_{\text{key}(\mathcal{U})}(\mathbf{R}_\mathcal{A})$ **do**
    $D(\mathcal{U}, t) \longleftarrow {}^\uparrow D(\mathcal{T}_1, \pi_{\text{key}(\mathcal{T}_1)}t) \times \cdots \times {}^\uparrow D(\mathcal{T}_k, \pi_{\text{key}(\mathcal{T}_k)}t)$;
    add expression $D(\mathcal{U}, t)$ to R;
  end
end
if $\mathcal{T}$ is a forest $\mathcal{T}_1, \ldots, \mathcal{T}_k$ **then add** $D(\mathcal{T}, \langle\rangle) \longleftarrow {}^\uparrow D(\mathcal{T}_1, \langle\rangle) \times \cdots \times {}^\uparrow D(\mathcal{T}_k, \langle\rangle)$ to R;
set $D(\mathcal{T}, \langle\rangle)$ as the root of R;

for *all expressions* D *in* R, *bottom-up* **do**
  if $D = \bigcup_a \langle\mathcal{A}{:}a\rangle \times {}^\uparrow D_a$ **then**
    $D \longleftarrow \bigcup_{a:D_a\neq\emptyset}\langle\mathcal{A}{:}a\rangle \times {}^\uparrow D_a$;
  else if $D = {}^\uparrow D_1 \times \cdots \times {}^\uparrow D_k$ *where some* $D_i = \emptyset$ **then**
    $D \longleftarrow \emptyset$;
  end
end
return D;

$D(\mathbf{R}, \mathcal{U}, t)$ in R are precisely as defined by Definition 5.13, and all expressions $D(\mathbf{R}, \mathcal{T}_\mathcal{A}, t)$ in R contain as a subexpression the one defined by Definition 5.13: they are of the same form, except that the range of their union, $\pi_\mathcal{A}\sigma_{\text{key}(\mathcal{A})=t}(\mathbf{R}_\mathcal{A})$, may be larger than $\pi_\mathcal{A}\sigma_{\text{key}(\mathcal{A})=t}Q(\mathbf{D})$. This shows that $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ as per Definition 5.13 is a subgraph of the parse graph of R.

Next we prove that the set of expressions R before block 2 is a d-representation of the result $Q(\mathbf{D})$. First note that, by labelling each subexpression $D(\mathbf{R}, \mathcal{X}, t)$ with the schema of $\mathcal{X}$, R indeed becomes a d-representation over the schema of $Q(\mathbf{D})$ with root $D(\mathbf{R}, \mathcal{T}, \langle\rangle)$. By top-down induction over $\mathcal{T}$ it follows that, in the traversal of R, each expression $D(\mathbf{R}, \mathcal{X}, t)$ is multiplied by singletons of all attributes from anc($\mathcal{X}$), and that those from key($\mathcal{X}$) coincide with t. Each singleton $\langle\mathcal{A}{:}a\rangle$ in R is in some expression $D(\mathbf{R}, \mathcal{T}_\mathcal{A}, t)$, where by construction we have $t \times \langle A{:}a\rangle \in \mathbf{R}_\mathcal{A}$, and this singleton is multiplied by t in any tuple represented by R. Therefore, for any tuple d represented by R and any node $\mathcal{A}$, $\pi_{\text{key}(\mathcal{A})\cup\mathcal{A}} \in \mathbf{R}_\mathcal{A}$, and hence $[\![R]\!] \subseteq \bowtie_\mathcal{A} \mathbf{R}_\mathcal{A}$. Since each relation R_i has all its attributes included in some key($\mathcal{A}$) $\cup\, \mathcal{A}$ and hence is unrestricted in $\mathbf{R}_\mathcal{A}$, we can deduce that $[\![R]\!] \subseteq \bowtie_i \mathbf{R}_i = Q(\mathbf{D})$. Since R contains the d-representation $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ and both are of the same schema, it follows that $[\![R]\!] \supseteq [\![\mathcal{T}^\uparrow(Q(\mathbf{D}))]\!] = Q(\mathbf{D})$. Therefore $[\![R]\!] = Q(\mathbf{D})$.

Finally, we prove that, after block 2, R equals $\mathcal{T}^\uparrow(Q(\mathbf{D}))$. We have shown earlier that, before block 2, R contains $\mathcal{T}^\uparrow(Q(\mathbf{D}))$, but its unions $D(\mathbf{R}, \mathcal{T}_\mathcal{A}, t)$ may contain

additional terms. Since $[\![D]\!] = [\![\mathcal{T}^\uparrow(Q(\mathbf{D}))]\!]$, all these additional terms must represent the empty relation, otherwise they would contribute additional tuples to $[\![D]\!]$. Block 2 of Algorithm 1 removes exactly these terms and no others, so the resulting d-representation R is therefore equal to $\mathcal{T}^\uparrow(Q(\mathbf{D}))$. □

PROPOSITION 8.2. *For any equi-join query Q, its d-tree $\mathcal{T}^\uparrow$, and database $\mathbf{D}$, Algorithm 1 runs in time $O(|\mathbf{D}|^{s^\uparrow(\mathcal{T}^\uparrow)} \cdot \log|\mathbf{D}| \cdot poly(|Q|, |\mathcal{S}|))$.*

PROOF. The computation of each $\mathbf{R}_\mathcal{A}$ takes $O(|\mathbf{D}|^{\rho^*(Q_{\text{key}(\mathcal{A})\cup\mathcal{A}})}) = O(|\mathbf{D}|^{s^\uparrow(\mathcal{T}^\uparrow)})$ using a worst-case optimal join algorithm [Ngo et al. 2012] and the size of $\mathbf{R}_\mathcal{A}$ is also $O(|\mathbf{D}|^{\rho^*(Q_{\text{key}(\mathcal{A})\cup\mathcal{A}})}) = O(|\mathbf{D}|^{s^\uparrow(\mathcal{T}^\uparrow)})$. The group-by can then be implemented using a sort in time $O(|\mathbf{R}_\mathcal{A}| \cdot \log|\mathbf{R}_\mathcal{A}|) = O(|\mathbf{D}|^{s^\uparrow(\mathcal{T}^\uparrow)} \cdot \log|\mathbf{D}| \cdot s^\uparrow(\mathcal{T}^\uparrow))$, while all remaining processing takes time linear in $|\mathbf{R}_\mathcal{A}|$. If the d-representation is constructed as a parse graph, the lookup of each expression name in an associative map takes time logarithmic in the total number of expressions, which is $O(|\mathbf{R}_\mathcal{A}|)$, so the total time is still quasi-logarithmic in $|\mathbf{R}_\mathcal{A}|$. The normalisation procedure implemented in block 2 of the algorithm takes time linear in the result computed thus far, so does not increase the runtime complexity. □

For ease of analysis, in Algorithm 1 we abstract away the computation of the joins $\mathbf{R}_\mathcal{A} = Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}(\mathbf{D})$: we apply a known worst-case optimal algorithm and use the results to construct the d-representation $Q(\mathbf{D})$. The queries $Q_{\text{key}(\mathcal{A})\cup\mathcal{A}}$ for different nodes $\mathcal{A}$ can partially overlap, and it is possible to amalgamate the entire computation into a single multi-way merge-join, as done in Olteanu and Závodný [2012] for f-representations only. However, this optimisation cannot reduce the data complexity of the algorithm, but only the factor hidden in poly($|Q|, |\mathcal{S}|$).

8.2. Computing D-Representations of Conjunctive Query Results

The algorithm for equi-join queries can be extended to arbitrary conjunctive queries using d-tree extensions. Recall from Proposition 6.9 that any d-tree of a conjunctive query Q can be extended to a d-tree of the equi-join $\hat{Q}$ of Q.

PROPOSITION 8.3. *Given any conjunctive query Q, a d-tree $\mathcal{T}^\uparrow$ of Q and its extension $\hat{\mathcal{T}}^\uparrow$, and a database $\mathbf{D}$, we can compute $\mathcal{T}^\uparrow(Q(\mathbf{D}))$ in time $O(|\mathbf{D}|^{s^\uparrow(\hat{\mathcal{T}}^\uparrow)} \cdot \log|\mathbf{D}|)$ with respect to data complexity.*

PROOF. Using the extension d-tree $\hat{\mathcal{T}}^\uparrow$, which is a d-tree of the equi-join $\hat{Q}$, we can compute the d-representation $\hat{\mathcal{T}}^\uparrow(\hat{Q}(\mathbf{D}))$ in time $O(|\mathbf{D}|^{s^\uparrow(\hat{\mathcal{T}}^\uparrow)} \cdot \log|\mathbf{D}|)$ by Proposition 8.2. Since $\hat{\mathcal{T}}^\uparrow$ is an extension of $\mathcal{T}^\uparrow$, it contains additional non-head attributes in some nodes, and also additional subtrees and subforests consisting of non-head attributes only. With respect to $\mathcal{T}^\uparrow(Q(\mathbf{D}))$, the d-representation $\hat{\mathcal{T}}^\uparrow(\hat{Q}(\mathbf{D}))$ therefore contains additional singletons $\langle A{:}a\rangle$ for those non-head attributes A that are in a node with some head attribute, and additional expressions $E(\mathcal{X}, t)$ for those subtrees and subforests $\mathcal{X}$ consisting of non-head attributes only. Both the additional singletons and expressions can be removed from $\hat{\mathcal{T}}^\uparrow(\hat{Q}(\mathbf{D}))$ in time linear in its size, so the total runtime is still $O(|\mathbf{D}|^{s^\uparrow(\hat{\mathcal{T}}^\uparrow)} \cdot \log|\mathbf{D}|)$ with respect to data complexity. □

9. SUCCINCTNESS GAP AND TREE DECOMPOSITIONS

In this final section we compare and quantify the succinctness of flat relational representations, f-representations over f-trees, and d-representations over d-trees in representing equi-join query results. We draw a complete picture of how succinct these three representation classes can be relative to each other: how much can relations be

$$1 \leq \underbrace{s^{\uparrow}(Q) = \mathrm{fhw}(Q) \overset{(1)}{\leq} \mathrm{fhpw}(Q) \overset{(2)}{\leq} s(Q)}_{\text{factor } O(\log |\mathcal{S}|)} \overset{(3)}{\leq} \rho^*(Q) \overset{(4)}{\leq} |Q|$$

Fig. 3. The hierarchy of parameters for nonempty equi-join queries Q. Each inequality may express a gap asymptotically as large as permitted by the remaining constraints. In particular, inequalities (1) and (2) may express a gap with a factor of $\Omega(\log |\mathcal{S}|)$, and inequalities (3) and (4) a factor of $\Omega(|Q|)$.

compacted by factorisation, and how much extra succinctness is brought by subexpression sharing in d-representations.

The succinctness of these representation systems for query results is characterised by the parameters $\rho^*(Q)$, $s(Q)$, and $s^{\uparrow}(Q)$ of the asymptotic size bounds introduced in Section 7. Recall that for a given equi-join query Q, $\rho^*(Q)$, $s(Q)$ and $s^{\uparrow}(Q)$ are the smallest numbers such that, for any database $\mathbf{D}$, the result $Q(\mathbf{D})$ has:

—a flat representation of size $O(|\mathbf{D}|^{\rho^*(Q)})$;
—an f-representation over an f-tree with size $O(|\mathbf{D}|^{s(Q)})$; and
—a d-representation over a d-tree with size $O(|\mathbf{D}|^{s^{\uparrow}(Q)})$.

We study the relationships of the parameters $\rho^*(Q)$, $s(Q)$, and $s^{\uparrow}(Q)$ to each other and to known parameters of fractional hypertree width and fractional hyperpath width.

We first show that d-trees are closely related to tree decompositions and that the parameter $s^{\uparrow}(Q)$ equals the fractional hypertree width $\mathrm{fhw}(Q)$ of Q (Corollary 9.4). Similarly but to a smaller extent, f-trees are related to path decompositions, and the parameter $s(Q)$ is greater than or equal to the fractional hyperpath width $\mathrm{fhpw}(Q)$ of Q (Corollary 9.9). Together with the trivial observation that $s(Q) \leq \rho^*(Q)$, we obtain the hierarchy of inequalities of parameters summarised in Figure 3.

We also quantify the gaps between these parameters. The parameter $s(Q)$ is bounded above by $O(\mathrm{fhw}(Q) \cdot \log |\mathcal{S}|)$, where $\mathcal{S}$ is the schema of Q (Proposition 9.12), and this bound is tight: we exhibit a class of queries with $s(Q) = \Omega(\mathrm{fhpw}(Q) \cdot \log |\mathcal{S}|)$ (Propositions 9.17 and 9.21), and from known results on path width it is also easy to exhibit queries with $\mathrm{fhpw}(Q) = \Omega(\mathrm{fhw}(Q) \cdot \log |\mathcal{S}|)$ (Proposition 9.15). The gap between $s(Q)$ and $\rho^*(Q)$ can also be as large as the hierarchy allows; we construct classes of queries with $s(Q) = 1$ while $\rho^*(Q) = |Q|$ (Proposition 9.22). Finally, we note that there exist arbitrarily large queries for which all mentioned parameters are $O(1)$, and queries for which all parameters are $\Omega(|Q|)$ (Proposition 9.23). These results are also summarised in Figure 3.

In this section we restrict our attention to those equi-join queries whose structure is precisely captured by their hypergraphs. Equi-join queries are also the traditional domain of structural decomposition methods, where the notions of fractional hypertree decompositions and fractional edge covers relate to size bounds and complexity of evaluation.

9.1. D-Trees and Tree Decompositions

There is a close connection between d-trees and fractional hypertree decompositions of the query hypergraph [Grohe and Marx 2006] for equi-join queries. We show how any d-tree $\mathcal{T}^{\uparrow}$ of an equi-join query Q can be translated into a fractional hypertree decomposition of Q with width $w = s^{\uparrow}(\mathcal{T}^{\uparrow})$, and any width-$w$ fractional hypertree decomposition of Q can be translated into a d-tree $\mathcal{T}^{\uparrow}$ with $s^{\uparrow}(\mathcal{T}^{\uparrow}) \leq w$. This implies that $s^{\uparrow}(Q)$ coincides with the fractional hypertree width of Q.

Let us first recall the definition of a fractional hypertree decomposition of a hypergraph.[3]

Definition 9.1 [*Grohe and Marx 2006*]. Let H be a hypergraph. A *tree decomposition* of H is a pair $(T, (B_t)_{t\in V(T)})$, where:

—T is a tree; and
—$(B_t)_{t\in V(T)}$ is a family of sets of vertices of H, called *bags*, such that each edge of H is contained in some B_t and, for each vertex v of H, the set $\{t : B_t \ni v\}$ is connected in T.

A *fractional hypertree decomposition* of H is a triple $(T, (B_t)_{t\in V(T)}, (\gamma_t)_{t\in V(T)})$, where:

—$(T, (B_t))$ is a tree decomposition; and
—$(\gamma_t)_{t\in V(T)}$ is a family of *weight functions* $E(H) \mapsto [0, \infty)$ such that, for each $t \in V(T)$, γ_t covers all vertices of B_t, that is, $\sum_{e\ni v} \gamma_t(e) \geq 1$ for all $v \in B_t$.

The weight of a weight function γ_t is $\text{weight}(\gamma_t) = \sum_{e\in E(H)} \gamma_t(e)$ and the width of the decomposition is $\max_{t\in V(T)} \text{weight}(\gamma_t)$. The *fractional hypertree width* of H, $\text{fhw}(H)$, is the minimum possible width of a fractional hypertree decomposition of H.

In a fractional hypertree decomposition of a hypergraph H, each weight function γ_t must be a fractional edge cover of the hypergraph H restricted to the vertices of B_t. Since we are primarily interested in fractional hypertree decompositions of minimum possible width, for a given tree decomposition $(T, (B_t))$ we often consider each γ_t to be an optimal fractional edge cover of B_t, and hence obtain a minimum width extension of $(T, (B_t))$ into a fractional hypertree decomposition $(T, (B_t), (\gamma_t))$. By the *fractional width* of a tree decomposition we mean the width of its minimal fractional extension; note that $\text{fhw}(H)$ is the minimal possible fractional width of a tree decomposition of H.

Next we show how any d-tree of an equi-join query Q corresponds to a tree decomposition of Q and vice versa. Intuitively, the vertices of the d-tree correspond to the vertices of the tree decomposition, and the sets $\text{key}(\mathcal{A}) \cup \mathcal{A}$ correspond to the bags $B_\mathcal{A}$. An example tree decomposition translated into a d-tree is depicted in Figure 4. Our translation ensures that the fractional width of the corresponding tree decomposition is at most the cost $s^\uparrow(\mathcal{T}^\uparrow)$ of the original d-tree $\mathcal{T}^\uparrow$ and vice versa.

PROPOSITION 9.2. *Let $\mathcal{T}^\uparrow$ be a d-tree of an equi-join query Q. There exists a fractional hypertree decomposition of Q with width $w = s^\uparrow(\mathcal{T}^\uparrow)$.*

PROOF. Let Q be an equi-join query and let $\mathcal{T}^\uparrow$ be a d-tree of Q. Consider the pair $(\mathcal{T}, (B_\mathcal{A})_{\mathcal{A}\in V(\mathcal{T})})$, where $\mathcal{T}$ is the underlying f-tree of $\mathcal{T}^\uparrow$ and the bag $B_\mathcal{A}$ contains the nodes of $\text{key}(\mathcal{A}) \cup \mathcal{A}$ for each node $\mathcal{A}$ of $\mathcal{T}$. We show that it is a tree decomposition of the query Q, with fractional width $s^\uparrow(\mathcal{T}^\uparrow)$.

First we show that each hyperedge of the query Q is contained in some bag $B_\mathcal{B}$. For any relation R of the query Q, the attributes of R lie on a root-to-leaf path in the f-tree $\mathcal{T}$ by the path condition of Proposition 6.1. For the lowest node $\mathcal{B}$ containing an attribute of R, all attributes of R are contained in $\text{path}(\mathcal{B})$. By Definition 6.4 characterising the d-trees of Q, all attributes of R must in fact lie in $\text{key}(\mathcal{B}) \cup \mathcal{B} \subseteq \bigcup B_\mathcal{B}$. Thus the hyperedge corresponding to the relation R is contained in the bag $B_\mathcal{B}$.

Next we show that for any node $\mathcal{B}$ of the query Q the set $\{\mathcal{A} : B_\mathcal{A} \ni \mathcal{B}\}$ is connected in $\mathcal{T}$. Since $\text{key}(\mathcal{A}) \cup \mathcal{A} \subseteq \text{anc}(\mathcal{A}) \cup \mathcal{A}$ for any $\mathcal{A}$, the node $\mathcal{B}$ may only be in $B_\mathcal{A}$ if $\mathcal{B}$ is an ancestor of $\mathcal{A}$ or equal to $\mathcal{A}$, or, equivalently, only if $\mathcal{A} \in \mathcal{T}_\mathcal{B}$. Also, by Definition 5.12,

[3]In this section we speak of a query and its hypergraph interchangeably; by a fractional hypertree decomposition of a query we mean the fractional hypertree decomposition of its hypergraph.

$\text{key}(\mathcal{A}) \subseteq \text{key}(\text{parent}(\mathcal{A})) \cup \text{parent}(\mathcal{A})$, so $\text{key}(\mathcal{A}) \cup \mathcal{A} \subseteq \text{key}(\text{parent}(\mathcal{A})) \cup \text{parent}(\mathcal{A}) \cup \mathcal{A}$ and hence $B_{\mathcal{A}} \subseteq B_{\text{parent}(\mathcal{A})} \cup \{\mathcal{A}\}$, for any node $\mathcal{A}$. Thus, if $B_{\mathcal{C}}$ does not contain $\mathcal{B}$ for some $\mathcal{C} \in \mathcal{T}_{\mathcal{B}}$, then $B_{\mathcal{D}}$ will not contain $\mathcal{C}$ for any $\mathcal{D}$ under $\mathcal{C}$. This shows that the set $\{\mathcal{A} : B_{\mathcal{A}} \ni \mathcal{B}\}$ is a connected subset of $\mathcal{T}$ (in fact, a connected subset of $\mathcal{T}_{\mathcal{B}}$), and concludes the proof that $(\mathcal{T}, (B_{\mathcal{A}})_{\mathcal{A} \in V(\mathcal{T})})$ is a tree decomposition of Q.

Finally, each bag $B_{\mathcal{A}}$ consists of the nodes of $\text{key}(\mathcal{A}) \cup \mathcal{A}$, so the cost of the optimal fractional edge cover $\gamma_{\mathcal{A}}$ of $B_{\mathcal{A}}$ is $\rho^*(Q_{\text{key}(\mathcal{A}) \cup \mathcal{A}})$, and the width of the corresponding fractional hypertree decomposition is exactly $\max_{\mathcal{A} \in V(Q)}(\rho^*(Q_{\text{key}(\mathcal{A}) \cup \mathcal{A}})) = s^{\uparrow}(\mathcal{T}^{\uparrow})$. □

PROPOSITION 9.3. *If there exists a fractional hypertree decomposition of an equi-join query Q with width w, then there exists a d-tree $\mathcal{T}^{\uparrow}$ of Q such that $s^{\uparrow}(\mathcal{T}^{\uparrow}) \leq w$.*

PROOF. Let $(T, (B_t), (\gamma_t))$ be a fractional hypertree decomposition of an equi-join query Q. Each bag B_t is a set of vertices of (the hypergraph of) Q, that is, equivalence classes of attributes under the selection condition of Q. We construct that d-tree $\mathcal{T}^{\uparrow}$ whose nodes are the vertices of Q by mimicking the structure of T. While each node may occur in multiple bags of T, in $\mathcal{T}$ we include each node only once, at its topmost occurrence in T. The formal definition of $\mathcal{T}^{\uparrow}$ follows.

Construction. For each bag B_t, let $B'_t \subseteq B_t$ be the set of those vertices which are not contained in B_a for any ancestor a of t. Chain the vertices of B'_t into a path, and construct an f-tree $\mathcal{T}$ by replacing each t in T by the path B'_t (all in-edges to t now enter the first node of B'_t and all out-edges from t now exit the last node of B'_t). Each vertex $\mathcal{A}$ lies in a connected subset of bags B_t, so there is exactly one B'_t containing $\mathcal{A}$ and hence exactly one occurrence of $\mathcal{A}$ in $\mathcal{T}$. Finally, construct $\mathcal{T}^{\uparrow}$ by annotating each node $\mathcal{A}$ of $\mathcal{T}$ with $\text{key}(\mathcal{A}) = \text{anc}(\mathcal{A}) \cap \bigcup B_t$, where t is such that $\mathcal{A} \in B'_t$.

Correctness. First we prove that the f-tree $\mathcal{T}$ is an f-tree of Q. Let A and B be attributes of a relation R, let a and b be such that $\mathcal{A} \in B'_a$ and $\mathcal{B} \in B'_b$, and let t be such that the hyperedge corresponding to R is a subset of B_t, so that $\mathcal{A}, \mathcal{B} \in B_t$. Then both a and b are ancestors of t in T, and hence a and b lie on a root-to-leaf path in T. This implies that $\mathcal{A}$ and $\mathcal{B}$ lie on a root-to-leaf path in $\mathcal{T}$, and shows that the path condition is satisfied.

Next we prove that the d-tree $\mathcal{T}^{\uparrow}$ is a d-tree of Q, that is, that for any $\mathcal{A}$, the nodes in the subtree $\mathcal{T}_{\mathcal{A}}$ can only depend on the vertices from $\text{key}(\mathcal{A})$. Suppose that some node $\mathcal{C}$ from $\mathcal{T}_{\mathcal{A}}$ depends on some $\mathcal{B}$ from $\text{anc}(\mathcal{A})$. Let a, b, c be such that $\mathcal{A} \in B'_a$, $\mathcal{B} \in B'_b$, and $\mathcal{C} \in B'_c$. Since $\mathcal{A}$ is an ancestor of $\mathcal{C}$ or $\mathcal{A} = \mathcal{C}$, a is an ancestor of c or $a = c$. In any case, if $\mathcal{C} \in B_t$ then t is a descendant of a or $t = a$. Since $\mathcal{B}$ and $\mathcal{C}$ are dependent, they share a hyperedge of Q, and hence there exists an r such that $\mathcal{B}, \mathcal{C} \in B_r$. By the preceding, r must be a descendant of a or $r = a$. Since $\mathcal{B} \in B'_b \subseteq B_b$ where b is an ancestor of a or equals a, and since the set $\{t : \mathcal{B} \in B_t\}$ is connected in T, we must also have $\mathcal{B} \in B_a$, that is, $\mathcal{B} \subseteq \bigcup B_a$. Thus $\mathcal{B} \subseteq \text{anc}(\mathcal{A}) \cap \bigcup B_a = \text{key}(\mathcal{A})$, as required.

Finally, for each node $\mathcal{A}$, the set of attributes $\text{key}(\mathcal{A}) \cup \mathcal{A}$ is contained in some $\bigcup B_t$, so $\rho^*(Q_{\text{key}(\mathcal{A}) \cup \mathcal{A}}) \leq \rho^*(Q_{\bigcup B_t}) \leq w$ where w is the width of the original fractional hypertree decomposition. It follows that $s^{\uparrow}(\mathcal{T}^{\uparrow}) = \max_{\mathcal{A}} \rho^*(Q_{\text{key}(\mathcal{A}) \cup \mathcal{A}}) \leq w$. □

The two-way correspondence yields the following equality.

COROLLARY 9.4 (PROPOSITIONS 9.2 AND 9.3). *For any equi-join query Q, $s^{\uparrow}(Q) = \text{fhw}(Q)$.*

PROOF. Let $\mathcal{T}^{\uparrow}$ be an optimal d-tree for the equi-join query Q. By Proposition 9.2, $\text{fhw}(Q) \leq s^{\uparrow}(\mathcal{T}^{\uparrow}) = s^{\uparrow}(Q)$. By Proposition 9.3, there exists a d-tree $\mathcal{T}^{\uparrow}$ such that $s^{\uparrow}(\mathcal{T}^{\uparrow}) = \text{fhw}(Q)$, so $s^{\uparrow}(Q) \leq \text{fhw}(Q)$. The result follows. □

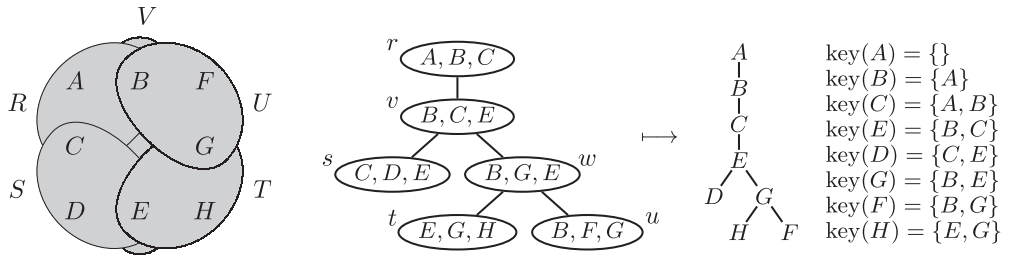


Fig. 4. Left to right: The hypergraph of query Q_3 from Example 9.5, its tree decomposition of fractional width $\frac{3}{2}$, and a corresponding d-tree $\mathcal{T}^{\uparrow}{}_{Q_3}$ with $s^{\uparrow}(\mathcal{T}^{\uparrow}{}_{Q_3}) = \frac{3}{2}$ as constructed in Proposition 9.3.

Example 9.5. We illustrate the correspondence between tree decompositions and d-trees on the query

$$Q_3 = R(A, B, C) \bowtie S(C, D, E) \bowtie T(E, G, H) \bowtie U(B, F, G) \bowtie V(B, E),$$

for clarity written here as a natural join. The hypergraph of Q_3 has vertices $A, B, \ldots, H$ and edges R, S, T, U, V, as depicted in Figure 4, left. A tree decomposition T_{Q_3} of Q_3 with vertices r, s, t, u, v, w is depicted in Figure 4, middle, where bags are shown in place of the vertices. The bags B_r, B_s, B_t, and B_u can each be covered by a single hyperedge R, S, T, and U, respectively. The bag $B_v = \{B, C, E\}$ can be covered by assigning weight $\frac{1}{2}$ to each of the hyperedges R, S, and V, and the bag $B_w = \{B, G, E\}$ by assigning weight $\frac{1}{2}$ to each of T, U, V. The fractional width of T_{Q_3} is thus $\frac{3}{2}$. The corresponding d-tree $\mathcal{T}^{\uparrow}{}_{Q_3}$ constructed by Proposition 9.3 is depicted in Figure 4, right: the nodes of Q are arranged into a tree by their topmost occurrence in the tree decomposition T_{Q_3} and, for each node N, the set key(N) contains those ancestors of N which are in the same bag as the topmost occurrence of N.

By applying Proposition 9.2 to $\mathcal{T}^{\uparrow}{}_{Q_3}$, we obtain a tree decomposition of Q_3 with fractional width $\frac{3}{2}$. It is not equal to the original decomposition T_{Q_3}; it contains an additional vertex A with $B_A = \{A\}$, its child B with $B_B = \{A, B\}$, and then a copy of T_{Q_3} as a subtree under B (not depicted here).

9.2. F-Trees and Path Decompositions

We draw a connection between f-trees and fractional hyperpath decompositions of equi-join queries, although a looser one than between d-trees and fractional hypertree decompositions. Any f-tree $\mathcal{T}$ of an equi-join query Q can be translated into a fractional hyperpath decomposition of the hypergraph of Q with width $s(\mathcal{T})$, and any width-w fractional hypertree decomposition of Q can be translated into an f-tree $\mathcal{T}$ with $s(\mathcal{T}) \leq w \cdot \log |\mathcal{S}|$, where $\mathcal{S}$ is the schema of Q. It follows that $s(Q)$ is greater than or equal to the fractional hyperpath width of Q, but can be greater by at most a factor logarithmic in $|\mathcal{S}|$. The next section shows that this logarithmic gap cannot be shrunk.

Definition 9.6. A *path decomposition* of a hypergraph H is a tree decomposition $(T, (B_t))$ of H for which the tree T is a path. A *fractional hyperpath decomposition* of H is a fractional hypertree decomposition $(T, (B_t), (\gamma_t))$ for which T is a path. The *fractional hyperpath width* of a hypergraph H, fhpw(H), is the minimum possible width of a fractional hyperpath decomposition of H.

Since any fractional hyperpath decomposition is also a fractional hypertree decomposition, fhw(H) $\leq$ fhpw(H) for any hypergraph H.

Next we show how any f-tree $\mathcal{T}$ of an equi-join query can be translated into a path decomposition of fractional width $s(\mathcal{T})$. Intuitively, each root-to-leaf path in $\mathcal{T}$

corresponds to a bag of the decomposition, and these bags are arranged into a path using some ordering of the leaves of $\mathcal{T}$.

Example 9.7. Consider query Q_3 from Example 9.5 and its f-tree $\mathcal{T}_{Q_3}$ as depicted in Figure 4. The path decomposition of Q_3 corresponding to this f-tree has bags $B_1 = \{A, B, C, E, D\}$, $B_2 = \{A, B, C, E, G, H\}$, and $B_3 = \{A, B, C, E, G, F\}$, in this order.

The translation is formalised in the following result.

PROPOSITION 9.8. *Let $\mathcal{T}$ be an f-tree of an equi-join query Q. There exists a path decomposition of Q with width $w = s(\mathcal{T})$.*

PROOF. *Construction.* Consider a left-to-right order of the nodes in the f-tree $\mathcal{T}$ induced by any left-to-right order of the children under each node. Let $L_1, \ldots, L_k$ be the leaves of $\mathcal{T}$ in this order, let B_i be the set of nodes of $\mathcal{T}$ (vertices of the hypergraph of Q) on the path from L_i to the root of $\mathcal{T}$, and let P be the path $1 - 2 - \cdots - k$.

Correctness. We show that $(P, (B_i))$ is a path decomposition of H. For each node $\mathcal{A}$, $\mathcal{A} \in B_i$ iff L_i is in the subtree $\mathcal{T}_\mathcal{A}$, and hence the set of indices i for which B_i contains $\mathcal{A}$ is a contiguous range of integers, that is, a connected subset of P. Moreover, for any relation R in Q, the attributes of R lie on a root-to-leaf path in $\mathcal{T}$, so the corresponding nodes are contained in B_i for some index i.

Finally, let γ_i be an optimal fractional edge cover of B_i. Then $(P, (B_i), (\gamma_i))$ is a fractional hyperpath decomposition of Q, and its width is $w = \max_i \rho^*(B_i) = \max_i \rho^*(Q_{\text{path}(L_i)})$. For any non-leaf vertex $\mathcal{A}$ of $\mathcal{T}$, there exists a leaf L_i under $\mathcal{A}$ and $\rho^*(Q_{\text{path}(\mathcal{A})}) \leq \rho^*(Q_{\text{path}(L_i)})$, so in fact $w = \max_\mathcal{A} \rho^*(Q_{\text{path}(\mathcal{A})}) = s(\mathcal{T})$. □

COROLLARY 9.9 (PROPOSITION 9.8). *For any equi-join query Q, $s(Q) \geq \text{fhpw}(Q)$.*

The translation of Proposition 9.8 cannot always be reversed. The path decompositions produced by the translation have a special property that, for any pair of nodes u and v, the sets $\{t : B_t \ni u\}$ and $\{t : B_t \ni v\}$ are either disjoint or one is contained in the other. A general path decomposition of Q does not have this property and cannot be translated back to an f-tree $\mathcal{T}$ of Q with $s(\mathcal{T})$ equal to the width of the decomposition. However, there exists a reverse translation for which $s(\mathcal{T})$ is by at most a logarithmic factor larger than the width of the original path decomposition. Moreover, such a translation can also be defined for arbitrary tree decompositions of Q.

Intuitively, the reverse translation works as follows. We pick a vertex V of the tree decomposition whose removal breaks the tree into smallest possible components. We recursively build an f-tree from each of these resulting components of the tree decomposition, and make them children subtrees of a path built from attributes in V. It is possible to prove that each path in the resulting f-tree contains attributes from only logarithmically many vertices of the original tree decomposition.

To establish this claim formally, we first prove an auxiliary lemma on balanced tree section and then the main result.

LEMMA 9.10. *For any tree T there exists a vertex v such that all connected components of $T \setminus v$ have at most $|V(T)|/2$ vertices.*

PROOF. Let v be a vertex of T for which the largest connected component of $T \setminus v$ has minimum possible number of vertices. For the sake of contradiction, suppose that $T \setminus v$ has a component C with more than $|V(T)|/2$ vertices, and let c be the vertex in C adjacent to c. Then the sets $C \setminus c$ and $T \setminus C$ are disconnected in $T \setminus c$, and have at most $|V(C)| - 1$ and $|V(T)|/2$ vertices, respectively. Therefore the largest connected component of $T \setminus c$ has less than $|V(C)|$ vertices, a contradiction. □

PROPOSITION 9.11. *If there exists a fractional hypertree decomposition* $(T, (B_t), (\gamma_t))$ *of* Q *with width* w*, then there exists an f-tree* $\mathcal{T}$ *of* Q *such that* $s(\mathcal{T}) \leq w \cdot (\log_2 |V(T)| + 1)$.

PROOF. Our construction is related to a known proof that any forest has logarithmic path width [Korach and Solel 1993]. For any tree decomposition $(T, (B_t))$ of Q, we rearrange the nodes of its underlying tree to attain height $\log |V(T)|$ (possibly losing the tree decomposition property), and then translate it into an f-tree $\mathcal{T}$ in which each root-to-leaf path will consist of at most $\log |V(T)|$ bags of the tree decomposition.

Construction. Let $(T, (B_t), (\gamma_t))$ be a fractional hypertree decomposition of Q. Construct a rooted tree balance(T) recursively as follows. Let v be a vertex in T such that all connected components of T have at most $|V(T)|/2$ vertices, and let $T_1, \ldots, T_k$ be the connected components of $T \setminus v$. Then balance(T) is a tree with root v and children subtrees balance$(T_1), \ldots,$ balance(T_k).

Next we repeat on balance(T) the construction of an f-tree from a tree of bags, used in the proof of Proposition 9.3. For each bag B_t, let B'_t be the set of vertices in B_t but not in B_a for any ancestor a of t in balance(T). Chain the vertices of each B'_t into a path, and construct $\mathcal{T}$ by replacing each B_t in balance(T) by the path B'_t.

Correctness. In the tree balance(T) the bags containing a given vertex $\mathcal{A}$ possibly do not form a connected subtree, but the following argument by contradiction shows that there is still only one occurrence of $\mathcal{A}$ in $\mathcal{T}$. If $\mathcal{A} \in B'_x$ and $\mathcal{A} \in B'_y$, then neither x nor y is an ancestor of the other, so they have a least common ancestor p different from x and y, and $\mathcal{A} \notin B_p$. The subtree of balance(T) rooted at p was constructed as balance(T_p) for some connected subtree T_p of T. Since x and y lie in different children subtrees of p in balance(T_p), they are in different connected components of $T_p \setminus p$. Therefore the set $\{t : B_t \ni \mathcal{A}\}$, containing x and y but not p, is disconnected in T_p and hence also in T. This contradicts $(T, (B_t))$ being a tree decomposition.

Next we show that $\mathcal{T}$ satisfies the path condition. Let A and B be attributes of a relation R, let a and b be such that $\mathcal{A} \in B'_a$ and $\mathcal{B} \in B'_b$. There exists a bag B_t containing all vertices of the hyperedge corresponding to R, in particular, $\mathcal{A}, \mathcal{B} \in B_t$. Then both a and b are ancestors of t in balance(T), and hence $\mathcal{A}$ and $\mathcal{B}$ lie on a root-to-leaf path in $\mathcal{T}$. This completes the proof that $\mathcal{T}$ is an f-tree of Q.

By induction we prove that depth(balance(T)) $\leq 1 + \log_2 |V(T)|$: if $|V(T)| = 1$ then depth(balance(T)) $= 1$ and if $|V(T)| > 1$ then depth(balance(T)) $= 1 +$ $\max_k$ depth(balance(T_k)) $\leq 1 + 1 + \log_2 \lfloor |V(T)|/2 \rfloor \leq 1 + \log_2 |V(T)|$.

Finally we prove the bound on $s(\mathcal{T})$. For any attribute A, if $\mathcal{A} \in B'_a$, then path$(\mathcal{A})$ in the tree $\mathcal{T}$ is contained in the labels of vertices of $\bigcup_{t \in \text{path}(a)} B_t$, where by path$(a)$ we mean the set containing a and the ancestors of a in balance(T). The weight function $\gamma_A = \sum_{t \in \text{path}(a)} \gamma_t$ covers all vertices in $\bigcup_{t \in \text{path}(a)} B_t$, the weight of each γ_t is at most w, and the size of path(a) is at most $1 + \log_2 |V(T)|$, so $\rho^*(Q_{\text{path}(\mathcal{A})}) \leq w \cdot (1 + \log_2 |V(T)|)$. Since this holds for any attribute A of Q, we also have $s(\mathcal{T}) \leq w \cdot (1 + \log_2 |V(T)|)$. □

PROPOSITION 9.12. *For equi-join queries* Q*, we have* $s(Q) = O(\text{fhw}(Q) \cdot \log |\mathcal{S}|)$.

PROOF. Let $\mathcal{T}^\uparrow$ be an optimal d-tree of Q. The proof of Proposition 9.2 constructs a fractional hypertree decomposition of Q with width $s^\uparrow(\mathcal{T}^\uparrow) = s^\uparrow(Q) = \text{fhw}(Q)$ such that the underlying tree T has $|V(Q)| \leq |\mathcal{S}|$ vertices. The result follows by Proposition 9.11. □

COROLLARY 9.13 (PROPOSITION 9.12). *For equi-join queries* Q*, we have* $s(Q) = O(s^\uparrow(Q) \cdot \log |\mathcal{S}|)$.

Example 9.14. Consider the chain query of 6 relations

$$Q_6 = \sigma_{B_1=A_2 \wedge B_2=A_3 \wedge \cdots \wedge B_5=A_6}(R_1(A_1, B_1) \times R_2(A_2, B_2) \times \cdots \times R_6(A_6, B_6))$$

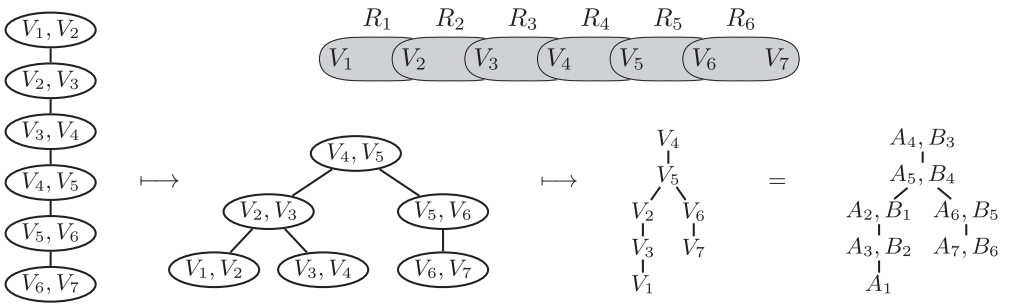


Fig. 5. Top right: Hypergraph of the query Q_6 from Example 9.14. Bottom left to right: A width-1 tree decomposition P (which is also a path decomposition) of the query Q_6, a rearrangement of its bags balance(P), and the resulting f-tree $\mathcal{T}$ as constructed in Proposition 9.11.

same as defined in Definition 9.16. Its hypergraph is a path of vertices $V_1 = A_1$, $V_2 = \{B_1, A_2\}$, ..., $V_6 = \{B_5, A_6\}$, $V_7 = \{B_6\}$ depicted in Figure 5, top, and has a straightforward width-1 path decomposition P with bags $\{V_1, V_2\}, \{V_2, V_3\}, \ldots, \{V_6, V_7\}$, depicted in Figure 5, left. This path decomposition can never result as a translation of an f-tree using Proposition 9.8, since, for instance, the sets of bags containing V_2 and V_3 are not disjoint, nor one contained in the other.

Proposition 9.11 translates any tree decomposition into an f-tree $\mathcal{T}$ with $s(\mathcal{T})$ only by a logarithmic factor larger than the width of the tree decomposition. For the tree decomposition P, the translation first constructs the tree balance(P) by repeatedly picking out the middle vertex as a root, and then the f-tree $\mathcal{T}$ by keeping the topmost occurrence of each node and removing others, as shown in Figure 5. The rearrangement $P \mapsto$ balance(P) ensures that each root-to-leaf path in the resulting f-tree $\mathcal{T}$ only contains vertices from a logarithmic number of bags from the original decomposition.

Note that as the constructed f-tree $\mathcal{T}$ is not necessarily optimal, we have $s(\mathcal{T}) = 3$ but $s(Q_6) = 2$ as witnessed by the complete binary f-tree $V_4(V_2(V_1, V_3), V_6(V_5, V_7))$.

9.3. Succinctness Gap for D-Representations

We show that the logarithmic upper bound on the gap between $s^\uparrow(Q)$ and $s(Q)$ is tight by exhibiting a class of queries for which $s(Q) = \Omega(s^\uparrow(Q) \cdot \log |\mathcal{S}|)$. First we show that the logarithmic gap exists between fhw(Q) and fhpw(Q), which implies the gap between $s^\uparrow(Q)$ and $s(Q)$ since

$$s^\uparrow(Q) = \text{fhw}(Q) \leq \text{fhpw}(Q) \leq s(Q).$$

Then we also exhibit a class of queries with a logarithmic gap between fhpw(Q) and $s(Q)$, that is, for which $s(Q) = \Omega(\text{fhpw}(Q) \cdot \log |\mathcal{S}|)$.

The gap between fhw(Q) and fhpw(Q) follows easily from existing results on tree width and path width.

PROPOSITION 9.15. *There exist arbitrarily large equi-join queries for which* $\text{fhpw}(Q) = \Omega(\text{fhw}(Q) \cdot \log |\mathcal{S}|)$.

PROOF. The complete binary tree T_h of height h has $2^h - 1$ vertices, tree width 1, and path width $\Omega(h)$ [Cattell et al. 1996]. Path width $\Omega(h)$ implies that any path decomposition of T_h has a bag B with $\Omega(h)$ vertices. Since T_h is a graph and all edges of T_h contain two vertices, two times the weight $\sum_e \gamma(e)$ of any weight function γ on B equals the sum of weights of all vertices $\sum_v \sum_{e \ni v} \gamma(e)$, which is at least $|B|$ if γ covers B. It follows that the weight of γ is $\Omega(h)$ and hence the fractional hyperpath width of T_h is $\Omega(h)$. The fractional hypertree width of T_h is still 1. Therefore the query Q_h^T

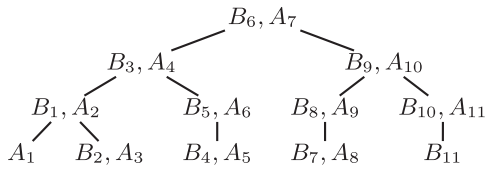


Fig. 6. The f-tree $\mathcal{T}_{11}$ for Q_{11} with height$(\mathcal{T}_{11}) = 4$ and $s(\mathcal{T}_{11}) = 3$ (from Example 9.18).

whose hypergraph is the tree T_h has fhw$(Q) = 1$ but fhpw$(Q) = \Omega(h) = \Omega(\log|2^h - 1|) = \Omega(\log|\mathcal{S}|)$. □

In the remainder of this section we show the gap between fhpw(Q) and $s(Q)$. A prototypical example for this gap is that of the chain queries.

Definition 9.16. Consider the relations R_i over schemas $\{A_i, B_i\}$ for $i \in \mathbb{N}$. For any natural number n we define the *chain query* Q_n to be the chain of $n-1$ joins

$$Q_n = \sigma_{B_1=A_2 \wedge B_2=A_3 \wedge \ldots \wedge B_{n-1}=A_n}(R_1 \times \cdots \times R_n).$$

The hypergraph of Q_n is a simple path of $n+1$ vertices denoted as $V_1 = \{A_1\}$, $V_2 = \{B_1, A_2\}, \ldots, V_n = \{B_{n-1}, A_n\}, V_{n+1} = \{B_n\}$ connected by the n edges $\{V_i, V_{i+1}\}$ corresponding to the relations R_i for $i = 1, \ldots, n$.

PROPOSITION 9.17. *For any chain query Q_n,* fhpw$(Q_n) =$ fhw$(Q_n) = s^{\uparrow}(Q_n) = 1$.

PROOF. Let P be the path $1 - 2 - \cdots - n$ and for each $i = 1, \ldots, n$ define the bag $G_i = \{V_i, V_{i+1}\}$ and the weight function γ_i as $R_i \mapsto 1$ and $R_j \mapsto 0$ for $j \neq i$. Then each hyperedge R_i of Q_n is contained in the bag G_i, for each vertex V_i of Q_n the set of j such that $G_j \ni V_i$ is connected, and each γ_i covers its corresponding bag G_i. Therefore $(P, (G_i)_{i=1}^n, (\gamma_i)_{i=1}^n)$ is a fractional path decomposition of Q_n. The weight of each γ_i is 1, so the weight of the decomposition is 1. Since Q_n is nonempty, any fractional hypertree decomposition has weight at least 1. It follows that fhpw$(Q_n) =$ fhw$(Q_n) = 1$ and, by Proposition 9.4, also $s^{\uparrow}(Q_n) = 1$. □

Proposition 9.11 bounds $s(Q_n)$ from above by $s(Q_n) \leq \log_2(n+1) + 1$. This bound is also witnessed by the balanced f-tree $\mathcal{T}_n$ constructed by picking the node $V_{\lfloor n/2 \rfloor + 1}$ in the middle of the chain query as a root and constructing its two children subtrees by recursively using the two resulting halves of the query: we definitely have $s(\mathcal{T}_n) \leq$ depth$(\mathcal{T}_n) = \lfloor \log_2(n+1) \rfloor + 1$.

Example 9.18. The f-tree $\mathcal{T}_{11}$ for the chain query Q_{11} (i.e., $n = 11$) is shown in Figure 6. Its depth is $\lfloor \log_2 12 \rfloor + 1 = 4$, so definitely $s(\mathcal{T}_{11}) \leq 4$. In fact $s(\mathcal{T}) = 3$, as $\rho^*(Q_{\text{path}(A_1)}) = 3$ and $\rho^*(Q_{\text{path}(A_i)}) \leq 3$ for all other A_i (where $Q_{\text{path}(A_i)}$ is the restriction of Q_{11} to path(A_i)).

The d-tree $\mathcal{T}^{\uparrow}$ that is a path of nodes $V_1 = \{A_1\}$, $V_2 = \{B_1, A_2\}, \ldots, V_{11} = \{B_{10}, A_{11}\}$ and $V_{12} = \{B_{11}\}$, rooted at V_1, with key$(V_{i+1}) = V_i$ for each i, is a valid d-tree of Q_{11}. Since each key$(V_{i+1}) \cup V_{i+1}$ is covered by the relation R_i, we have $s^{\uparrow}(\mathcal{T}^{\uparrow}) = 1$ and hence also $s^{\uparrow}(Q_{11}) = 1$.

Next we prove that, up to a constant factor, $\mathcal{T}_n$ is optimal for Q_n and thus the bound of Proposition 9.11 is tight. We first prove a lemma limiting those f-trees among which we need to search for an optimal f-tree. It states that under any node of an f-tree it always pays off to branch into the maximal possible number of branches.

Definition 9.19. An f-tree $\mathcal{T}$ of an equi-join query Q is *maximally branching* if, for each node $\mathcal{A}$, the children subtrees of $\mathcal{A}$ correspond to the connected components of the query $Q_{\mathcal{T}_{\mathcal{A}} \setminus \mathcal{A}}$.

Note that, for any f-tree $\mathcal{T}$ of an equi-join query and any node $\mathcal{A}$, each of the connected components of $Q_{\mathcal{T}_\mathcal{A}\setminus\mathcal{A}}$ is wholly contained in one of the children subtrees of $\mathcal{A}$. Otherwise, some relation of that connected component would have its attributes in two distinct children subtrees of $\mathcal{A}$, which would violate the path condition. An f-tree $\mathcal{T}$ is maximally branching if the vertices of each connected component of $Q_{\mathcal{T}_\mathcal{A}\setminus\mathcal{A}}$ form a different subtree under $\mathcal{A}$.

LEMMA 9.20. *For any equi-join query Q, there exists a maximally branching f-tree $\mathcal{T}_b$ with $s(\mathcal{T}_b) = s(Q)$.*

PROOF. Let $\mathcal{T}$ be any f-tree of Q with $s(\mathcal{T}) = s(Q)$. Construct $\mathcal{T}_b$ by splitting $\mathcal{T}$ as much as possible but reflecting the original hierarchy of the nodes in $\mathcal{T}$. Formally, define split($\mathcal{T}$) recursively as follows. For a forest $\mathcal{U}$, split($\mathcal{U}$) $= \bigcup_{\text{tree } \mathcal{T} \text{ in } \mathcal{U}}$ split($\mathcal{T}$). For a tree $\mathcal{T}$ with root $\mathcal{A}$ and forest of children subtrees $\mathcal{U}$, let $\{\mathcal{T}_1, \ldots, \mathcal{T}_k\} =$ split($\mathcal{U}$), let $A = \{i : \mathcal{T}_i \text{ depends on } \mathcal{A}\}$, let $\mathcal{T}_a$ be a tree with root $\mathcal{A}$ and children subtrees $\{\mathcal{T}_i\}_{i\in A}$, and define split($\mathcal{T}$) $= \{\mathcal{T}_i\}_{i\notin A} \cup \mathcal{T}_a$.

By structural induction on $\mathcal{T}$, we can prove that split($\mathcal{T}$) satisfies the path constraint, so $\mathcal{T}_b =$ split($\mathcal{T}$) is an f-tree of Q. By construction, $\mathcal{T}_b$ is maximally branching. Also, by structural induction we prove that $Q_{\text{path}_{\text{split}(\mathcal{T})}(\mathcal{A})} \subseteq Q_{\text{path}_\mathcal{T}(\mathcal{A})}$ for any node $\mathcal{A}$ in $\mathcal{T}$, so $\rho^*(Q_{\text{path}_{\text{split}(\mathcal{T})}(\mathcal{A})}) \leq \rho^*(Q_{\text{path}_\mathcal{T}(\mathcal{A})})$ and hence $s(\mathcal{T}_b) \leq s(\mathcal{T})$. Since $\mathcal{T}$ is optimal, we must in fact have $s(\mathcal{T}_b) = s(\mathcal{T}) = s(Q)$. □

PROPOSITION 9.21. *For any chain query Q_n, $s(Q_n) = \Omega(\log n)$.*

PROOF. Let $\mathcal{T}$ be a maximally branching optimal f-tree of Q_n. By top-down induction on $\mathcal{T}$ we can prove that the vertices of any subtree $\mathcal{T}_\mathcal{A}$ of $\mathcal{T}$ are $\{V_i\}_{i\in I}$ for a contiguous interval I of integers, and hence that any node in $\mathcal{T}$ has at most two children. It follows that the height of $\mathcal{T}$ is at least $\log_2(n+1)+1$. Since each hyperedge of Q_n covers at most two vertices, $s(Q_n) = s(\mathcal{T}) > (\log_2(n+1)+1)/2 = \Omega(\log n)$. □

9.4. Succinctness Gap for F-Representations

For nonempty equi-join queries, any f-representation of the query result must be at least linear in the database size, while the result size can be exponential in the query size. We show that there exist queries for which this size gap is attained.

PROPOSITION 9.22. *There exist arbitrarily large equi-join queries Q such that $s^\uparrow(Q) = s(Q) = 1$ and $\rho^*(Q) = |Q|$.*

PROOF. The product query $Q = R_1 \times \cdots \times R_n$ over unary relations $R_1, \ldots, R_n$ has $\rho^*(Q) = n = |Q|$ but $s(Q) = 1$. □

The product query is a trivial example, but there exist many others. In particular, any equi-join query Q, in which at least one attribute per relation is not involved in joins, has $\rho^*(Q) = |Q|$, yet many such queries still retain small $s(Q)$. For example, queries Q whose Boolean projections $\pi_\emptyset Q$ are hierarchical [Dalvi and Suciu 2007] admit an f-tree $\mathcal{T}$ with $s(\mathcal{T}) = 1$. For each root-to-leaf path in such an f-tree there is a relation with attributes in each node of the path. A simple example of a hierarchical query is the join $\sigma_{A_1=\cdots=A_n}(R_1 \times \cdots \times R_n)$, where each R_i is over a schema $\{A_i, B_i\}$.

On the other hand, there exist queries for which $s(Q) = \rho^*(Q)$, and whose results hence cannot benefit from factorisations over f-trees. This happens when no branching is possible in f-trees of Q and all f-trees of Q are paths, so that $Q_{\text{path}(\mathcal{B})} = Q$ for the bottom node $\mathcal{B}$. All f-trees of a query Q are paths iff any two nodes are dependent, that is, any two attribute classes have attributes from a common relation.

PROPOSITION 9.23. *There exist arbitrarily large equi-join queries Q such that $s^\uparrow(Q) = s(Q) = \rho^*(Q) = \Omega(|\mathcal{S}|)$.*

PROOF. Consider the relations $R_{i,j}$ for $1 \leq i < j \leq n$ with schemas $\{A^i_{i,j}, A^j_{i,j}\}$. Let $Q = \sigma_\psi(\times_{i<j} R_{i,j})$, where ψ equates all attributes with the same superscript. The hypergraph of Q is the complete graph on n nodes, so the possible f-trees of Q are the $n!$ paths of these nodes and the possible d-trees have $\text{key}(N) = \text{anc}(N)$ for all nodes N.

For each such d-tree $\mathcal{T}^\uparrow$, the query $Q_{\text{key}(\mathcal{B})\cup\mathcal{B}} = Q_{\text{path}(\mathcal{B})}$ of the bottom node $\mathcal{B}$ includes all nodes of $\mathcal{T}^\uparrow$ and hence is equal to Q, and its fractional edge cover number is $\rho^*(Q_{\text{key}(\mathcal{B})\cup\mathcal{B}}) = \rho^*(Q) = \binom{n}{2}\frac{1}{n-1} = \frac{n}{2}$. (An optimal fractional edge cover assigns weight $\frac{1}{n-1}$ to each of the $\binom{n}{2}$ relations.) It follows that $s^\uparrow(\mathcal{T}^\uparrow) = s(\mathcal{T}) = \rho^*(Q) = \frac{n}{2}$ for any f-tree $\mathcal{T}$ of Q, and hence $s^\uparrow(Q) = s(Q) = \rho^*(Q) = \frac{n}{2} = \Omega(|\mathcal{S}|)$. □

10. DIRECTIONS FOR FUTURE WORK

This work introduces the parameters $s(Q)$ and $s^\uparrow(Q)$ which characterise the succinctness of f-representations and d-representations of conjunctive query results, and relates these parameters to other known measures such as fractional hypertree width. The complexity of computing $s(Q)$ and $s^\uparrow(Q)$ for a given conjunctive query Q is unaddressed and still open. The related complexity of computing the fractional hypertree width is also open, with partial results on its approximation [Marx 2010].

The algorithms for computing f-representations and d-representations of query results presented in this work are proven worst-case optimal. Recent results [Ngo et al. 2013] towards instance-optimal join algorithms with flat relational results can perhaps be extended to the factorised case.

For arbitrary input relations beyond query results, the computation of an optimal factorised representation is likely hard, similar to the Σ_2^p-hardness of minimisation of Boolean functions [Buchfuhrer and Umans 2008]. Determining the precise complexity of various flavours of problems of finding minimal factorisations, as well as quantifying the succinctness gaps between various flavours of f-representations and d-representations, is subject to future work. A robust approach to approximate instance-based factorisation would be desirable in practice.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers, whose suggestions helped improve the presentation of this article.

REFERENCES

S. Abiteboul and N. Bidoit. 1986. Non first normal form relations: An algebra allowing data restructuring. *J. Comput. Syst. Sci.* 33, 3, 361–393.

S. Abiteboul, R. Hull, and V. Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

S. Agrawal, V. Narasayya, and B. Yang. 2004. Integrating vertical and horizontal partitioning into automated physical database design. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'04)*. 359–370.

A. Atserias, M. Grohe, and D. Marx. 2008. Size bounds and query plans for relational joins. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS'08)*. 739–748.

G. Bagan, A. Durand, and E. Grandjean. 2007. On acyclic conjunctive queries and constant delay enumeration. In *Proceedings of the 21st International Workshop on Computer Science Logic (CSL'07)*. Lecture Notes in Computer Science, vol. 4646. Springer, 208–222.

N. Bakibayev, T. Kočiský, D. Olteanu, and J. Závodný. 2013. Aggregation and ordering in factorized databases. *Proc. VLDB Endow.* 6, 14, 1990–2001.

N. Bakibayev, D. Olteanu, and J. Závodný. 2012. FDB: A query engine for factorised relational databases. *Proc. VLDB Endow.* 5, 11, 1232–1243.

F. Bancilhon, P. Richard, and M. Scholl. 1982. On line processing of compacted relations. In *Proceedings of the 8th International Conference on Very Large Data Bases (VLDB'82)*. 263–269.

D. S. Batory. 1979. On searching transposed files. *ACM Trans. Database Syst.* 4, 4, 531–544.

P. A. Boncz, S. Manegold, and M. L. Kersten. 1999. Database architecture optimized for the new bottleneck: Memory access. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB'99)*. 54–65.

R. K. Brayton. 1987. Factoring logic functions. *IBM J. Res. Devel.* 31, 2, 187–198.

D. Buchfuhrer and C. Umans. 2008. The complexity of Boolean formula minimization. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP'08)*. 24–35.

K. Cattell, M. J. Dinneen, and M. R. Fellows. 1996. A simple linear-time algorithm for finding path-decompositions of small width. *Inf. Process. Lett.* 57, 4, 197–203.

L. Cerf, J. Besson, C. Robardet, and J.-F. Boulicaut. 2009. Closed patterns meet n-ary relations. *ACM Trans. Knowl. Discov. Data* 3, 1, 3.

H. Chen and M. Grohe. 2010. Constraint satisfaction with succinctly specified relations. *J. Comput. Syst. Sci.* 76, 8, 847–860.

P. Cudré-Mauroux, E. Wu, and S. Madden. 2009. The case for Rodentstore: An adaptive, declarative storage system. In *Proceedings of the 4th Biennial Conference on Innovative Data Systems Research (CIDR'09)*.

N. Dalvi and D. Suciu. 2007. Efficient query evaluation on probabilistic databases. *VLDB J.* 16, 4, 523–544.

C. Delobel. 1978. Normalization and hierarchical dependencies in the relational data model. *ACM Trans. Database Syst.* 3, 3, 201–222.

F. Geerts, B. Goethals, and T. Mielikäinen. 2004. Tiling databases. In *Proceedings of the 7th International Conference on Discovery Science (DS'04)*. Lecture Notes in Computer Science, vol. 3245, Springer, 278–289.

G. Gottlob. 2012. On minimal constraint networks. *Artif. Intell.* 191-192, 42–60.

G. Gottlob, N. Leone, and F. Scarcello. 2000. A comparison of structural CSP decomposition methods. *Artif. Intell.* 124, 2, 243–282.

T. J. Green, G. Karvounarakis, and V. Tannen. 2007. Provenance semirings. In *Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'07)*. 31–40.

M. Grohe and D. Marx. 2006. Constraint solving via fractional edge covers. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm (SODA'06)*. 289–298.

M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden. 2010. HYRISE—A main memory hybrid storage engine. *Proc. VLDB Endow.* 4, 2, 105–116.

F. Henglein and K. F. Larsen. 2010. Generic multiset programming with discrimination-based joins and symbolic Cartesian products. *Higher-Order Symbol. Comput.* 23, 3, 337–370.

T. Imielinski, S. Naqvi, and K. Vadaparty. 1991. Incomplete object-A data model for design and planning applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'91)*. 288–297.

G. Jaeschke and H. J. Schek. 1982. Remarks on the algebra of non first normal form relations. In *Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS'82)*. 124–138.

W. Kent. 1983. A simple guide to five normal forms in relational database theory. *Comm. ACM* 26, 2, 120–125.

E. Korach and N. Solel. 1993. Tree-width, path-width, and cutwidth. *Discr. Appl. Math.* 43, 1, 97–101.

A. Makinouchi. 1977. A consideration on normal form of not-necessarily-normalized relation in the relational data model. In *Proceedings of the 3rd International Conference on Very Large Data Bases (VLDB'77)*. Vol. 3. 447–453.

D. Marx. 2010. Approximating fractional hypertree width. *ACM Trans. Algor.* 6, 2, 29:1–29:17.

H. Q. Ngo, D. T. Nguyen, C. Ré, and A. Rudra. 2013. Towards instance optimal join algorithms for data in indexes. http://arxiv.org/abs/1302.0914.

H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. 2012. Worst-case optimal join algorithms (extended abstract). In *Proceedings of the 31st Symposium on Principles of Database Systems (PODS'12)*. 37–48.

D. Olteanu and J. Huang. 2008. Using OBDDs for efficient query evaluation on probabilistic databases. In *Proceedings of the 2nd International Conference on Scalable Uncertainty Management (SUM'08)*. 326–340.

D. Olteanu, C. Koch, and L. Antova. 2007. World-set decompositions: Expressiveness and efficient algorithms. In *Proceedings of the 11th International Conference on Database Theory (ICDT'07)*. 194–208.

D. Olteanu and J. Závodný. 2011. On factorisation of provenance polynomials. In *Proceedings of the 3rd USENIX Workshop on the Theory and Practice of Provenance.*

D. Olteanu and J. Závodný. 2012. Factorised representations of query results: Size bounds and readability. In *Proceedings of the 15th International Conference on Database Theory (ICDT'12).* 285–298.

Z. M. Ozsoyoglu and L.-Y. Yuan. 1987. A new normal form for nested relations. *ACM Trans. Database Syst.* 12, 1, 111–136.

J. Pearl. 1989. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Fransisco.

S. Rendle. 2013. Scaling factorization machines to relational data. *Proc. VLDB Endow.* 6, 5, 337–348.

P. Sen, A. Deshpande, and L. Getoor. 2010. Read-once functions and query evaluation in probabilistic databases. *Proc. VLDB Endow.* 3, 1, 1068–1079.

J. Shute, R. Vingralek, B. Samwel, B. Handy, C. Whipkey, E. Rollins, M. Oancea, K. Little-Field, D. Menestrina, S. Ellner, J. Cieslewicz, I. Rae, T. Stancescu, and H. Apte. 2013. F1: A distributed SQL database that scales. *Proc. VLDB Endow.* 6, 11, 1068–1079.

M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. 2005. C-store: A column-oriented DBMS. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05).* 553–564.

T. L. Veldhuizen. 2014. Triejoin: A simple, worst-case optimal join algorithm. In *Proceedings of the 17th International Conference on Database Theory (ICDT'14).* 96–106.

Received July 2013; revised May 2014; accepted July 2014