Worst-Case Optimal Join at a Time

Radu Ciucanu[1] and Dan Olteanu[1]

1 **Department of Computer Science, University of Oxford**
{radu.ciucanu,dan.olteanu}@cs.ox.ac.uk

First version: November 2015. Latest version: March 2016

Abstract

Joins are at the core of database systems, yet worst-case optimal join algorithms have been developed only recently. At the outset of this effort is the observation that the standard join plans are suboptimal as their intermediate results may be larger than the final result. To attain worst-case optimality, new join algorithms are monolithic and thus avoid intermediate results.

The conceptual contribution of this paper is the observation that this monolithic recipe is an artefact of the tabular data representation and not necessary for optimality. Our technical contribution is an effective procedure that achieves optimality with multiway join-at-a-time query plans by employing succinct representations of the intermediate results and a new join operator called Joen that can work on such representations. We further study the optimality of join-at-a-time query plans across four data representation systems of increasing succinctness.

1998 ACM Subject Classification H.2.4 Systems: Query processing

Keywords and phrases join algorithms, data representation, worst-case optimality

1 Introduction

Joins are at the core of database systems, yet worst-case optimal join algorithms such as LeapFrog TrieJoin (LFTJ) [16], NPRR [10], and FDB [13] have been developed only recently. These algorithms exploit the observation that the standard join plans are suboptimal as their intermediate results may be larger than the final result [2]. To attain worst-case optimality, these algorithms are monolithic in that they avoid intermediate results [11].

The conceptual contribution of this paper is the observation that the aforementioned monolithic recipe for join computation is an artefact of the tabular representation of intermediate results and not necessary for optimality. Our technical contribution is a procedure that achieves (worst-case) optimality with multiway join-at-a-time query plans that create and work on succinct representations of the intermediate results. Such plans solve one join variable at a time. They differ from standard query plans that join one relation at a time.

We consider four factorized representation systems for intermediate and final results of join queries in relational databases and study the optimality of join-at-a-time query plans for all of them. They encode relations as algebraic expressions with data values, Cartesian product, and union. To factorize relations and avoid redundancy in their representation, they use the distributivity of Cartesian product over union and a mechanism to define repeating expressions and to use references to such definitions in place of their expressions. The representations in $\mathcal{T}$ are **t**ries that factor out data values occurring in several union terms, while those in $\mathcal{F}$ may **f**actor out arbitrary algebraic expressions. $\mathcal{E}$ consists of $\mathcal{F}$-representations with d**e**finitions for (sub)tries of input data, while the representations in $\mathcal{D}$ are factorizations with **d**efinitions for arbitrary factorizations.

$$\mathcal{D} \supsetneq \mathcal{E} \supsetneq \mathcal{F} \supsetneq \mathcal{T}$$

$$1 \leq \underbrace{\mathit{fhtw}(Q) \leq \overbrace{e(Q) \leq f(Q)}^{\text{up to } 1}}_{\text{up to } \log(|Q|)} \leq \underbrace{\rho^*(Q)}_{\text{up to } |Q|} \leq |Q|$$

Figure 1 Relationships between the four representation systems and their width measures.

Representation		*Final result*			
		$\mathcal{T}$	$\mathcal{F}$	$\mathcal{E}$	$\mathcal{D}$
Intermediate	$\mathcal{T}$	— [Prop. 13]	— [Prop. 13]	— [Prop. 13]	— [Prop. 13]
	$\mathcal{F}$	— [Prop. 13]	— [Prop. 13]	— [Prop. 13]	— [Prop. 13]
	$\mathcal{E}$	✔ [Cor. 20]	✔ [Cor. 20]	✔ [Cor. 20]	— [Prop. 16]
	$\mathcal{D}$	✔ [Cor. 20]	✔ [Cor. 20]	✔ [Cor. 20]	✔ [Cor. 20]

Figure 2 Map of worst-case optimality for join processing across four representation systems.

Figure 1 depicts the syntactic relationship between the four representation systems: $\mathcal{T}$-representations are (a strict subset of) $\mathcal{F}$-representations, which are $\mathcal{E}$-representations, which are $\mathcal{D}$-representations. Each $\mathcal{X}$-representation system has a width measure $w_{\mathcal{X}}(Q)$ to quantify the worst-case optimal size $O(|\mathbf{D}|^{w_{\mathcal{X}}(Q)})$ of the $\mathcal{X}$-representation of the query result $Q(\mathbf{D})$ for a join query Q and databases $\mathbf{D}$. This is the fractional edge cover number $\rho^*(Q)$ for $\mathcal{T}$-representations [2], the factorization width $f(Q)$ for $\mathcal{F}$-representations [12], the e-width $e(Q)$ for $\mathcal{E}$-representations, and the fractional hypertree width $\mathit{fhtw}(Q)$ [6] for $\mathcal{D}$-representations [13]. Figure 1 also depicts the relationship between these width measures.

The above tight bounds also hold for the time to compute the factorized query results (data complexity, modulo log factors) as $\mathcal{T}$-representations [10] and $\mathcal{F}/\mathcal{D}$-representations [13].

Figure 2 overviews the technical results of this paper, namely which of the representation systems support worst-case optimal join-at-a-time query plans. Optimality is achieved whenever the intermediate results are $\mathcal{D}$-representations regardless of the representation of the final results, or when the intermediate results are $\mathcal{E}$-representations but then the final results can be $\mathcal{T}/\mathcal{F}/\mathcal{E}$-representations and not $\mathcal{D}$-representations. We show that the $\mathcal{F}$-representations [4] cannot attain optimality. This motivates $\mathcal{E}$-representations, which are slightly more succinct than $\mathcal{F}$-representations and can attain optimality.

These results assume the input database given as $\mathcal{T}$-representation, which is our proxy for the standard tabular data representation. (We leave as future work the case where the input is given as $\mathcal{F}/\mathcal{E}/\mathcal{D}$-representation.) They rely on two complementary contributions.

(1) The query plans use a new join operator called *Joen* that works on factorized representations and takes time linear in the sizes of its input and output (modulo log factors).

(2) The factorized intermediate results of the query plans have sizes that are asymptotically upper bounded by the size of an optimal factorization of the final result. We call such plans output-bounded. The stricter notion of monotonically width-increasing query plans requires that for each plan step its output size asymptotically upper bounds its input size.

Organization. Section 2 highlights aspects of our contribution with an example from the literature [11] that has been originally used to show the suboptimality of standard query plans. Section 3 defines the factorized representation systems. Section 4 introduces join-at-a-time query plans, which are monotonically width-increasing (Section 5) and made up of

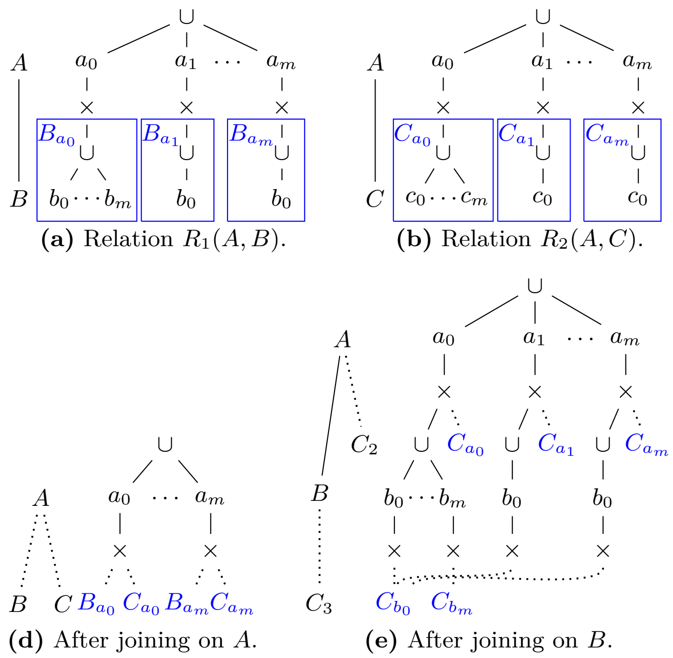


Figure 3 Join-at-a-time processing of the triangle query. First row: $\mathcal{T}$-representations of relations $R_1(A, B)$, $R_2(A, C)$, and $R_3(B, C)$ over the $\mathcal{T}$-paths to their left. The labeled boxes are definitions referenced from factorizations in the second row. Second row: $\mathcal{E}$-representations of intermediate results over $\mathcal{E}$-trees and the $\mathcal{T}$-representation for the final result over a $\mathcal{T}$-path. A dotted edge denotes a reference to an existing factorization fragment.

Joen steps (Section 6). Section 7 reviews related work. Section 8 highlights benefits of our approach. Appendix contains the proofs of formal statements and further examples.

2 Revisiting the Triangle Query

We show how to compute the triangle query $Q_\triangleleft = R_1(A, B), R_2(A, C), R_3(B, C)$ using a query plan that first joins on A, then on B, and finally on C. The input database consists of the three relations R_1, R_2, R_3 (Figure 2 in [11]):

$$R_1 = \{(a_0, b_0), \ldots, (a_0, b_m), (a_1, b_0), \ldots, (a_m, b_0)\}$$
$$R_2 = \{(a_0, c_0), \ldots, (a_0, c_m), (a_1, c_0), \ldots, (a_m, c_0)\}$$
$$R_3 = \{(b_0, c_0), \ldots, (b_0, c_m), (b_1, c_0), \ldots, (b_m, c_0)\}$$

Figure 3(a) depicts a $\mathcal{T}$-representation of R_1 that first groups by A and then by B. Its nesting structure is given by the linear order $A\{B\}$ of its variables, which we call a $\mathcal{T}$-path. The $\mathcal{T}$-representation lists a union of all sorted distinct A-values, and under each such value a it lists the union of all (sorted) B-values that occur together with a in R_1. This factorization of R_1 exploits the distributivity law of Cartesian product over union to avoid the repetition of a_0 with each of b_0 to b_m. The $\mathcal{T}$-representations of R_2 and R_3 are over the $\mathcal{T}$-paths $A\{C\}$ and respectively $B\{C\}$, cf. Figures 3(b-c). Each of these $\mathcal{T}$-representations has $3m+2$ values. The $\mathcal{T}$-representation of the query result in Figure 3(f) has $6m+3$ values.

We first compute the join on A to obtain the factorization J_1 in Figure 3(d): We intersect the two lists of A-values in the $\mathcal{T}$-representations of R_1 and R_2 and for each value a in the intersection we keep references to the corresponding unions of B-values in R_1 and of C-values in R_2. This is where our approach departs from the standard join evaluation: (1) We do not materialize the pairs of values for B and C for each value a. (2) We keep references to the unions of values for B and C from the input instead of copying them to J_1.

To accommodate (1), we allow factorizations with symbolic (non-materialized) Cartesian products of unions of values for B and C. The nesting structures of these factorizations are not anymore given by $\mathcal{T}$-paths, but by *factorization trees* or $\mathcal{F}$-trees. An $\mathcal{F}$-tree for J_1 is $A\{B, C\}$ and encodes the conditional independence of variables B and C given variable A.

To accommodate (2), we label the factorization fragments corresponding to the subtries corresponding to unions of B-values and C-values in the $\mathcal{T}$-representations of R_1 and respectively R_2, and we use references to them instead of their copies in J_1. Whereas $\mathcal{F}$-representations avoid the materialization of Cartesian products, $\mathcal{E}$-representations may also use definitions of subtries in the input data. This referencing mechanism is denoted in the nesting structure of J_1 by dotted edges: An $\mathcal{E}$*-tree*, such as the nesting structure of J_1, is thus an $\mathcal{F}$-tree with dotted edges to $\mathcal{T}$-paths. Figure 3(d) depicts J_1 and its $\mathcal{E}$-tree. Although not exemplified in this section, the most general nesting structures are that for factorized representations with arbitrary definitions: They are called $\mathcal{D}$*-trees* and extend $\mathcal{E}$-trees in that they may have dotted edges to other $\mathcal{D}$-trees and not only to $\mathcal{T}$-paths.

To create J_1, we need $m + 1$ computation steps to intersect the two (ordered) unions of A-values. If we were to copy the unions of B-values and C-values and create an $\mathcal{F}$-representation instead, we would need additional $2(m + 1) + 2m$ steps. Both cases would thus need linearly many computation steps. In contrast, $\mathcal{T}$-representations of J_1 must have sizes quadratic in m, since for a_0, $m+1$ different B-values would need to be paired with $m+1$ different C-values. This is where query plans with $\mathcal{T}$-representations for intermediate results become suboptimal: The $\mathcal{T}$-representation of the intermediate result J_1 has quadratic size, whereas the $\mathcal{T}$-representation of the final result of $Q_{\triangleleft}$ has linear size.

We next compute the join on B to obtain the factorization J_2 in Figure 3(e): We materialize the intersection of the union of B-values under each A-value in J_1 with the union of B-values in R_3, and we keep references to the unions of C-values. Since the variable C now appears twice in the nesting structure of J_2 due to both R_2 and R_3, we disambiguate its occurrences using the index of their input relations. Whereas referencing was not necessary to keep J_1's computation linear in m, it is now necessary for J_2 since C_{b_0} occurs under each A-value in J_2 and materializing all its occurrences would require space and time quadratic in m. Any $\mathcal{F}$-representation of J_2 would thus be of quadratic size and larger than the size of the final result, whereas its $\mathcal{E}$-representation stays linear. This shows the limitation of $\mathcal{F}$-representations over $\mathcal{E}$-representations.

We finally compute the join on C in J_2 to obtain the factorization of the query result in Figure 3(f): Under each A-value a and B-value under a, we intersect the unions of C_2-values and of C_3-values. The unions C_{b_0} and C_{a_0} are equal and their intersection requires $m + 1$ steps. The intersection for all other pairs of C_{b_i} and C_{a_j} takes constant time and yields c_0. This last join step takes space and time linear in m. For our class of input relations, we can thus compute the triangle query in time and space linear in m. For arbitrary input relations, the triangle query can be computed in $O(\sqrt{|R_1| \cdot |R_2| \cdot |R_3|})$, cf. Appendix A.

The nesting structures of the intermediate and final results of a join-at-a-time query plan as well as of the definitions used in their factorizations are defined by hypertree decompositions of the query hypergraph. They dictate the class of the representations and their asymptotic sizes. For instance, cyclic queries require $\mathcal{T}$-paths and their results do not factorize well with no asymptotic saving beyond tries. However, their non-cyclic subqueries may admit more succinct factorizations. The path query of length seven admits linear-size $\mathcal{D}$-representations for intermediate and final results, quadratic-size $\mathcal{E}$-representations for intermediate and final results, cubic-size $\mathcal{F}$-representations for the final result, and quartic-size $\mathcal{T}$-representations for the final result (Example B.3 in Section B.2).

3 Four Factorized Representation Systems

This section presents a unified framework for four factorized representations of results to join queries. Except for the $\mathcal{E}$-representation system, the development in this section has been previously introduced in the literature [13]. Due to space limitation, technical material on size measures for these systems is deferred to Appendix B.

We consider representations of relational data that are expressions in a relational algebra subset with union, Cartesian product, data values, and definitions (or named views). We call them factorized representations, since they use algebraic factorization based on the distributivity of product over union to reduce redundancy in data representation.

▶ **Definition 1** ([13])**.** A *factorized representation* is a list of expressions $(D_1, \ldots, D_n)$ where D_i can contain references to D_j for $j > i$ and is referenced at least once if $i > 1$. Such expressions are relational algebra expressions over a schema Σ and of the following forms:

- $\emptyset$, representing the empty relation over Σ,
- $\langle\rangle$, representing the relation consisting of the nullary tuple, if $\Sigma = \emptyset$,
- a, representing the relation with one tuple having one data value (a), if $\Sigma = \{A\}$ and the value $a \in \mathrm{Dom}(A)$,
- $(E_1 \cup \cdots \cup E_k)$, representing the union of the relations represented by E_i, where each E_i is an expression over Σ,
- $(E_1 \times \cdots \times E_k)$, representing the Cartesian product of the relations represented by E_i, where each E_i is an expression over schema Σ_i such that Σ is the disjoint union of all Σ_i.
- a reference ${}^{\uparrow}E$ to a definition of an expression E over Σ.

$\mathcal{D}$-representations are factorized representations. *$\mathcal{F}$-representations* are $\mathcal{D}$-representations without references. *$\mathcal{T}$-representations* are $\mathcal{F}$-representations where in each product $E_1 \times \cdots \times E_k$ all but at most one expression E_i are data values. *$\mathcal{E}$-representations* are $\mathcal{D}$-representations, where references are restricted to $\mathcal{T}$-representations.

Without loss of generality, we consider factorized representations with alternating unions and products; indeed, if one of the terms in a union (product) is again a union (product), we can flatten it out into a single union (product) of terms. For any $\mathcal{D}$-representation D consisting of expressions $\{D_1, \ldots, D_n\}$, we can start with the root expression D_1 and repeatedly replace the references ${}^{\uparrow}D_j$ by the expressions D_j until we obtain a single expression without references, which is an $\mathcal{F}$-representation. The $\mathcal{T}$-representations are tries of relations. They are our proxy for the standard tabular representation of relations.

Figures 3(a)-(c), (f) show $\mathcal{T}$-representations. Figures 3(d)-(e) show $\mathcal{E}$-representations. The definitions B_{a_i}, C_{a_j}, and C_{b_k} in these $\mathcal{E}$-representations are for $\mathcal{T}$-representations that are unions of data values. Figure 4 shows a $\mathcal{D}$-representation.

Although Definition 1 permits arbitrary factorized representations, we are interested in this paper in factorized results of join queries over nesting structures defined by the join hypergraph. Such nesting structures are orders on the query variables. Variable orders serve three purposes. They define the nesting structure of the factorized query results and thus the permitted factorizations. They guide our join algorithm: Query plans correspond to

Loop4=$R_1(A,B)$, $R_2(B,C)$, $R_3(C,D)$, $R_4(A,D)$
Database instances with $|R_1| = \ldots = |R_4| = 3n$:

R_1

A	B
a_0	b_1
$\ldots$	$\ldots$
a_0	b_n
a_1	b_0
$\ldots$	$\ldots$
a_{2n}	b_0

R_2

B	C
b_0	c_1
$\ldots$	$\ldots$
b_0	c_n
b_1	c_0
b_1	c_1
$\ldots$	$\ldots$
b_n	c_0
b_n	c_n

R_3

C	D
c_0	d_{2n+1}
$\ldots$	$\ldots$
c_0	d_{4n}
c_1	d_1
$\ldots$	$\ldots$
c_n	d_n

R_4

A	D
a_0	d_1
$\ldots$	$\ldots$
a_0	d_{3n}

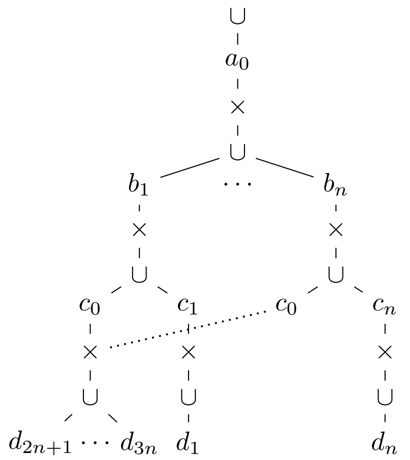


Figure 4 $\mathcal{D}$-representation of the result of query Loop4 on a class of database instances. While its size is $O(n)$, the size of $\mathcal{T}/\mathcal{F}/\mathcal{E}$-representations of the query result would be $O(n^2)$. This is because we would materialize the n D-values under each of the n B-values. The referencing capabilities of $\mathcal{E}$-representations are limited to $\mathcal{T}$-representations from input relations, hence they cannot refer to D-values that represent the intersection of lists of D-values from R_3 and R_4.

top-down traversals of variable orders, where for each variable we resolve all join conditions on the occurrences of that variable before considering the next variable. They define the size bounds and computation time for query results.

▶ **Definition 2** ([13]). Given a join query Q and two variables A and B in Q, A *depends* on B if they occur in the same relation symbol in Q. A *variable order* Δ for Q is a pair of a rooted forest with one node per variable in Q and a function key_Δ mapping each variable A to the subset of its ancestor variables in Δ on which the variables in the subtree rooted at A depend. When Δ is clear from context, we write $key(A)$ instead of $key_\Delta(A)$.

A variable order Δ for Q satisfies the following constraints:

- The variables of each relation symbol in Q lie along the same root-to-leaf path in Δ.
- For every variable B that is a child of a variable A, $key_\Delta(B) \subseteq key_\Delta(A) \cup \{A\}$.

We further define the key of an entire subtree or forest T of a variable order Δ as the union of keys of all variables in T: $key(T) = \bigcup_{A \in T} key_\Delta(A)$.

If two variables A and B are dependent on each other, then the choice for a value for A may restrict the choice for a value for B. If they are not dependent, we can represent the values for A separately from those for B instead of explicitly representing their Cartesian product. The succinctness of factorized representations lies in the exploitation of (in)dependency information, which is kept for each variable in its key. For a variable A, a tuple of values for the variables in $key(A)$ is called *context*. In a factorized representation E over a variable order Δ, the context ctx functionally determines the factorization fragment E_A rooted at A: $\eta_A[ctx] = E_A$, where η is a function.

A constructive definition of a factorized representation E over a variable order Δ of a query result R is as follows. We define $\Delta(R)$ to be the set of expressions $\eta_T[ctx]$ for all subtrees or forests T in Δ and all $ctx \in \pi_{key(T)}(R)$ as follows ($V_A = \pi_A \sigma_{key(A)=ctx} R$):

- For any leaf A in Δ, $\eta_A[ctx] = \bigcup_{a \in V_A} a$.
- For any subtree $\Delta_A = A\{T\}$, $\eta_{\Delta_A}[ctx] = \bigcup_{a \in V_A} a \times {}^{\uparrow}\eta_{\{T\}}[\pi_{key(\{T\})}(t \times a)]$.

- For any forest $\{T_1, \ldots, T_k\}$, $\eta_{\{\mathcal{T}_1,\ldots,\mathcal{T}_k\}}[ctx] = {}^{\uparrow}\eta_{T_1}[\pi_{key(T_1)} ctx] \times \cdots \times {}^{\uparrow}\eta_{T_k}[\pi_{key(T_k)} ctx]$.

If Δ is empty, then $\Delta(R) = \{\eta_\Delta[\langle\rangle]\}$, where $\eta_\Delta[\langle\rangle]$ is $\emptyset$ if $R = \emptyset$ and $\langle\rangle$ otherwise.

▶ **Example 3.** The factorization of the identity query R_1 in Figure 3(a) is over a path variable order $A\{B\}$ since we first group by A-values and then by B-values. Here, $key(B) = \{A\}$ and both variables lie on the only path in the variable order since they are dependent. The factorized join J_1 in Figure 3(d), which is the join of R_1 and R_2 on A, is over the tree variable order $A\{B, C\}$, where we first group by A and then under each A-value we branch out and group by B in one branch and by C in the other branch. Here, $key(B) = key(C) = \{A\}$, A and B lie along a path, and the same for A and C. The factorized join J_2 in Figure 3(e) is over the tree variable order $A\{B\{C_3\}, C_2\}$, where $key(C_3) = \{B\}$ and $key(B) = key(C_2) = \{A\}$. The variables A and C_3 are not dependent on each other, yet they both depend on B and lie on the same path with it.

The $\mathcal{D}$-representation for the result of the query Loop4 in Figure 4 is over the variable order $A\{B\{C\{D\}\}\}$, where $key(A) = \emptyset$, $key(B) = \{A\}$, $key(C) = \{A, B\}$, and $key(D) = \{A, C\}$. Even though A and C are not in the same relation, $A \in key(C)$ because $A \in key(D)$ and D is a child of C in the variable order. Then, $B \notin key(D)$ because D does not occur in the same relation with B and its children do not depend on B (since D has no children).

The last variable order for the query LW4 in Figure 6 is the same path $A\{B\{C\{D\}\}\}$, yet $key(D) = \{A, B, C\}$. In this case, $key(D)$ has all ancestors of D since D occurs with each of them in a relation in LW4. ◻

The four representation systems correspond to various restrictions of variable orders.

▶ **Definition 4.** $\mathcal{D}$*-trees* are the variable orders from Definition 2. $\mathcal{F}$*-trees* are $\mathcal{D}$-trees where the key for each variable is the set of all of its ancestors. $\mathcal{T}$*-paths* are $\mathcal{F}$-trees restricted to forests of paths. The $\mathcal{E}$*-trees* are $\mathcal{D}$-trees with the following restrictions: (1) If a variable A does not have all ancestors as key, then the variable order rooted at A is a $\mathcal{T}$-path. (2) The variables in this $\mathcal{T}$-path occur in the same relation symbol in the query.

The $\mathcal{T}$*-representation system* is the set of $\mathcal{T}$-representations over $\mathcal{T}$-paths. For $\mathcal{X} \in \{\mathcal{F}, \mathcal{E}, \mathcal{D}\}$, the $\mathcal{X}$*-representation system* is the set of $\mathcal{X}$-representations over $\mathcal{X}$-trees.

$\mathcal{D}$*-trees* are the most general variable orders considered in this paper. They are a different syntax for the fractional hypertree decompositions of the query hypergraph (there is a one-to-one mapping between $\mathcal{D}$-trees and hypertree decompositions of the join hypergraph) [13]. In case not all ancestors of a variable A are in the key of A in a variable order Δ, then in a $\mathcal{D}$-representation over Δ the same factorization fragments rooted at A-values may be repeated for every tuple of values for variables that are ancestors of A and not in $key(A)$. Here is where references come in handy: We define such factorization fragments and refer to them by their names instead of repeatedly copying them. To effectively use definitions, we thus need the key information in variable orders.

The $\mathcal{F}$-trees are the nesting structures of $\mathcal{F}$-representations or factorized databases [4]. In contrast to $\mathcal{D}$-trees, definitions cannot save repetitions of factorization fragments in $\mathcal{F}$-representations since a tuple t of values for the ancestor variables of A functionally determines the factorization fragment rooted at an A-value and in general a different fragment may occur under each distinct tuple t.

The $\mathcal{E}$*-trees* are more permissive than $\mathcal{F}$-trees and less permissive than $\mathcal{D}$-trees. They state that definitions can be used in $\mathcal{E}$-representations but only for $\mathcal{T}$-representations of (projections of) input relations and not for $\mathcal{D}$-representations of arbitrary join queries.

▶ **Example 5.** Figure 3 illustrates $\mathcal{T}$-paths (a)-(c) and $\mathcal{E}$-trees (d)-(e). Figure 6 shows $\mathcal{E}$-trees, while Figure 5 shows $\mathcal{D}$-trees (all except the last one are also $\mathcal{E}$-trees). We use dotted edges to depict the $\mathcal{T}$-path components in the $\mathcal{E}$-trees. ☐

Each representation system has a width measure thay captures the tight bounds on the size of representation for any join result, cf. Figure 1, discussion in Section 1, and Appendix B.2.

4 Join-at-a-time Query Plans

We evaluate a join query by compiling it into a query plan, such that each step of the plan executes all join conditions on the occurrences of one variable. The plan step is a multiway join that takes an input factorization to an output factorization in one of our representation systems. We show that this approach is worst-case optimal for certain combinations of representation systems for intermediate and final results, and it is not optimal for all others.

For the positive results, we derive plans where each step satisfies two properties (for every input database): (1) It takes time linear in the sizes of its input and output. (2) The size of its output asymptotically upper bounds the size of its input. The first property is ensured by a novel join algorithm called Joen, cf. Section 6. The second property is called monotonically width-increasing, cf. Section 5. Worst-case optimality of query processing then follows by taking query results of worst-case optimal sizes within different representation systems.

For the negative results, we exhibit examples of queries for which there are no output-bounded query plans. These positive and negative findings are summarized in Figure 2.

We next define our notion of query plans and give properties that are essential for their optimality. For a query Q, a variable order Δ_{in} for the input factorization, and a variable order Δ_{out} of m variables for factorized query result, a query plan for Q is a sequence of variable orders from Δ_{in} to Δ_{out}. To define these intermediate variable orders, we take any topological order τ of the variables in Δ_{out}. Then, the i-th step in the plan would be a multiway join on the i-th variable in τ; if there is no join condition on the i-th variable, then the i-th step is trivial as it does nothing. A query plan is thus uniquely defined by Δ_{in}, Δ_{out}, and τ. To make this intuition more precise, we need additional notions. Given a variable order Δ and any variable A in Δ, let $depth_\Delta(A)$ be the depth of A in Δ, where the root of Δ has depth 0 and the children of a variable at depth i have depth $i+1$.

▶ **Definition 6.** Given a variable A in a variable order Δ, we define the *assignment order* ω on the occurrences $(A^{(j)})_{j\in[s]}$ of A in Δ as follows ($\forall j_1, j_2 \in [s]$):

$$A^{(j_1)} <_\omega A^{(j_2)} \Leftrightarrow depth_\Delta(A^{(j_1)}) < depth_\Delta(A^{(j_2)}) \vee depth_\Delta(A^{(j_1)}) = depth_\Delta(A^{(j_2)}) \wedge j_1 < j_2.$$

An assignment order for the occurrences of a variable A corresponds to the order in which we find assignments of (unions of) values for these occurrences in a top-down traversal of the input factorization. The variable order of the result of the multijoin on A is called packing.

▶ **Definition 7.** Given a variable order Δ and a variable A with occurrences $(A^{(j)})_{j\in[s]}$ in Δ, the *packing of* Δ *on* A is the variable order Δ' that is Δ where: A_s becomes A; the subtrees of $(A^{(j)})_{j\in[s]}$ become the subtrees of A; and $(A^{(j)})_{j\in[s-1]}$ are removed.

We also say that A is *packed at* Δ'. We next define a query plan as a sequence of packing steps and later show how to compute the keys in the variable order after a packing step.

▶ **Definition 8.** Given a join query Q, a database $\mathbf{D}$ over the $\mathcal{T}$-path Δ_{in}, a variable order Δ_{out} of m variables for the query result $Q(\mathbf{D})$, and a topological order τ of Δ_{out}. A *plan*

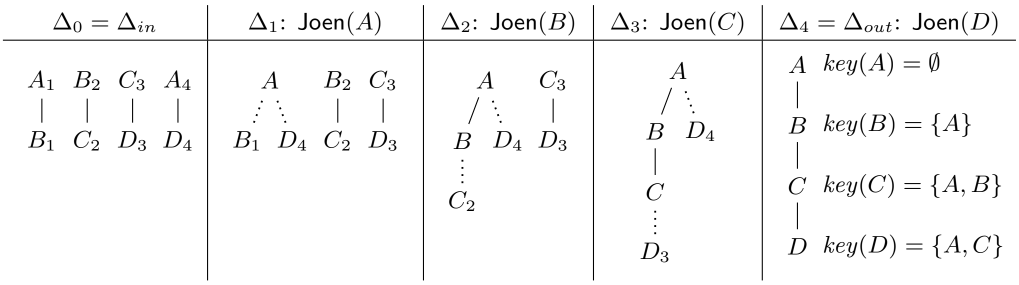


Figure 5 Query plan with $\mathcal{D}$-trees for Loop4 $= R_1(A, B), R_2(B, C), R_3(C, D), R_4(A, D)$.

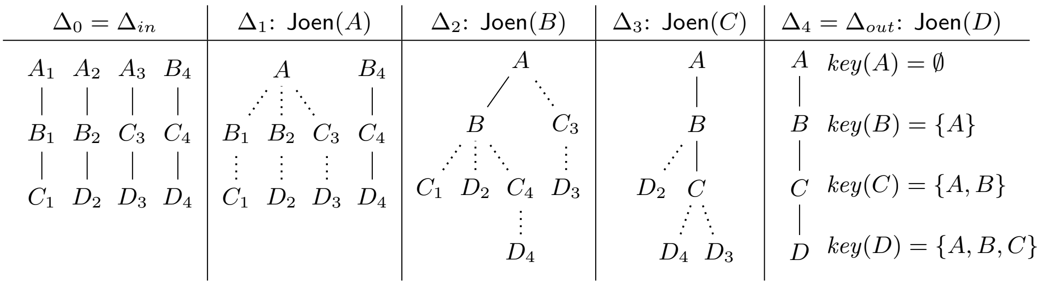


Figure 6 Query plan with $\mathcal{E}$-trees for LW4 $= R_1(A, B, C), R_2(A, B, D), R_3(A, C, D), R_4(B, C, D)$.

for Q is a sequence of variable orders $\Delta_0 = \Delta_{in}, \ldots, \Delta_m = \Delta_{out}$ such that $\forall i \in [m] : \Delta_i$ is the packing of Δ_{i-1} on the i-th variable in τ.

The plans in Definition 8 use a total order on the variables to be joined and resolve them one variable at a time. Further plans that use partial orders on the variables are possible, but we leave their investigation to future work.

▶ **Example 9.** Figures 5 and 6 show plans for a loop query of length four Loop4 and for the Loomis-Whitney query of length four LW4. For both queries, we pack the variables in the same order: A, B, C, D. The assignment orders for Loop4 are: (A_1, A_4), (B_2, B_1), (C_3, C_2), and (D_4, D_3) in Δ_0 to Δ_3. The assignment orders for LW4 are: (A_1, A_2, A_3), (B_4, B_1, B_2), (C_3, C_1, C_4), and (D_2, D_3, D_4) in in Δ_0 to Δ_3. For the triangle query in Figure 3, the assignment orders are: (A_1, A_2), (B_3, B_1), and (C_2, C_3) in the variable orders in Figures 3: (a)-(c), (d), and respectively (e). ◻

For any variable order Δ_i that is a packing of Δ_{i-1} on the i-th variable A in a query plan $(\Delta_0, \ldots, \Delta_m)$, the key $key_{\Delta_i}(X)$ of a variable X is $key_{\Delta_m}(X)$ if X is A and $key_{\Delta_{i-1}}(X)$ otherwise. More generally, our query plans enjoy two important properties: (1) preservation of variable keys across the plan steps and (2) one-lookahead path of variable occurrences.

▶ **Proposition 10.** *In a plan* $\Delta_0, \ldots, \Delta_m$ *over* m *variables, for any variable* A *it holds that:*

1. [Key preservation] $\quad \forall i \in [m] : key_{\Delta_{i-1}}(A) \subseteq key_{\Delta_i}(A)$.
   In particular, if A *is the variable packed at* Δ_i*, then* $\forall j \in [m] : key_{\Delta_j}(A) = key_{\Delta_m}(A)$.
2. [One-lookahead path] *Assume* A *is the variable packed at* Δ_i*. If any two of its occurrences have the same depth in* Δ_i*, then they are siblings or roots in* Δ_i*.*

The key preservation property holds since packing is the only change between consecutive steps in a plan, and, when a variable is packed, its key becomes the key from the final plan step and contains the union of the keys of its occurrences, since the result of the multiway join on these occurrences now depends on all variables that the individual occurrences depended on. The key for any other variable stays the same.

The one-lookahead path property says that, if we would have a virtual root of all variable orders in Δ_i, then the occurrences of variable A are along one root-to-leaf path in Δ_i or children of variables on that path. This holds by virtue of our choice for the initial variable order (we consider without loss of generality that whenever we are given a variable order Δ, the order of the attributes in each input $\mathcal{T}$-representation is compatible with Δ) and our construction of the intermediate variable orders: The first variable to join is root in input variable orders, and after joining on a variable A, the next variable to join has its occurrences as children of previously joined variables or as root in variable orders. Our join algorithm Joen relies on this property.

▶ **Example 11.** Consider the plan for query Loop4 in Figure 5 consisting of variable orders that are $\mathcal{D}$-trees. Once packed: variable A keeps $key_{\Delta_i}(A) = \emptyset$ for $1 \leq i \leq 4$; variable B keeps $key_{\Delta_i}(B) = \{A\}$ for $2 \leq i \leq 4$; variable C keeps $key_{\Delta_i}(C) = \{A, B\}$ for $3 \leq i \leq 4$; and variable D has $key_{\Delta_4}(D) = \{A, C\}$. At packing time, the key of each of A, B, and D becomes precisely the union of the keys of its occurrences. For variable C, however, $key_{\Delta_2}(C_3) = \emptyset$ and $key_{\Delta_2}(C_2) = \{B\}$ at packing time, yet $key_{\Delta_3}(C) = \{A, B\} \supset (key_{\Delta_2}(C_3) \cup key_{\Delta_2}(C_2))$. This is because after packing, C is placed above D, whose key contains A.

We verify the one-lookahead path property for Loop4: A_1 and A_2 are root in Δ_0; B_2 and B_1 are in unconnected components of Δ_1 with B_2 root, and the same for C_3 and C_2 in Δ_2 with C_3 root; D_3 lies along the path from A to D_3 and D_4 is a child of A in Δ_3.

For LW4 in Figure 6: A_1, A_2, A_3 are all root in Δ_0; B_1 and B_2 are siblings while B_4 is root in a different component of Δ_1; C_1 and C_4 are siblings and have A as ancestor, C_3 is a child of A and C_4 is a child of B along the path from A to C_1 in Δ_2; finally, D_2 and D_3 are children of variables along the path from A to D_4 in Δ_3 and D_3 and D_4 are siblings. ◻

5 Output-Bounded and Monotonically Width-Increasing Query Plans

To attain worst-case optimality, our query plans have to satisfy the following constraint: The sizes of the intermediate results are asymptotically upper bounded by the size of the final result. This is captured by the notion of output-bounded query plans:

▶ **Definition 12.** A query plan $(\Delta_0, \dots, \Delta_m)$ is *ouput-bounded* for the $\mathcal{X}$-representation system with width measure $w_{\mathcal{X}}$ if $\forall i \in [m] : w_{\mathcal{X}}(\Delta_{i-1}) \leq w_{\mathcal{X}}(\Delta_m)$.

This constraint is not satisfied by $\mathcal{T}/\mathcal{F}$-representations of intermediate and final results; the case of the $\mathcal{T}$-representation system follows immediately from the literature [2].

▶ **Proposition 13.** *The triangle query has no query plan that is output-bounded for the $\mathcal{T}/\mathcal{F}$-representation systems. This also holds for $\mathcal{F}$-representations of the intermediate results and $\mathcal{T}$-representations of the final result.*

This implies that we cannot attain worst-case optimality of join-at-a-time query processing using $\mathcal{T}/\mathcal{F}$-representations. In contrast, any join query admits query plans that are output-bounded for the $\mathcal{D}/\mathcal{E}$-representation systems in a stricter sense:

▶ **Definition 14.** A query plan $(\Delta_0, \dots, \Delta_m)$ is *monotonically width-increasing* for the $\mathcal{X}$-representation system with width measure $w_{\mathcal{X}}$ if $\forall i \in [m] : w_{\mathcal{X}}(\Delta_{i-1}) \leq w_{\mathcal{X}}(\Delta_i)$.

We consider two refinements of this property. First, the input database is given as a $\mathcal{T}$-representation, so Δ_0 is a $\mathcal{T}$-path and $w_{\mathcal{X}}(\Delta_0) = 1$ regardless of the representation system $\mathcal{X}$. Second, we may allow for a different representation system for Δ_m and thus for the final query result. This accommodates the common case of one representation system for both the input database and the query result, e.g., the $\mathcal{T}$-representation system, and a more succinct representation system for the intermediate results, e.g., the $\mathcal{E}$-representation system.

In contrast to the $\mathcal{F}$-representation system, the slightly more succinct $\mathcal{E}$-representation system (and thus also the exponentially more succinct $\mathcal{D}$-representation system) admits monotonically width-increasing query plans.

▶ **Theorem 15.** *Every join query has a query plan that is monotonically width-increasing for the $\mathcal{E}/\mathcal{D}$-representation systems. This also holds for: $\mathcal{E}$-representations of the intermediate results and $\mathcal{T}/\mathcal{F}$-representations of the final result; and for $\mathcal{D}$-representations of the intermediate results and $\mathcal{T}/\mathcal{F}/\mathcal{E}$-representations of the final result.*

Theorem 15 leaves out one negative case. Let the path query

$$P_7 = R_1(A, B), R_2(B, C), R_3(C, D), R_4(D, E), R_5(E, F), R_6(F, G), R_7(G, H).$$

▶ **Proposition 16.** *The Path7 query has no query plan that is output-bounded for $\mathcal{E}$-representations of the intermediate results and $\mathcal{D}$-representations of the final result.*

▶ **Example 17.** For the plan for query Loop4 in Figure 5 consisting of $\mathcal{D}$-trees we have: $fhtw(\Delta_0) = fhtw(\Delta_1) = fhtw(\Delta_2) = 1 < fhtw(\Delta_3) = fhtw(\Delta_4) = fhtw(\mathsf{Loop4}) = 2$.

For the plan for query LW4 in Figure 6 consisting of $\mathcal{E}$-trees we have: $e(\Delta_0) = e(\Delta_1) = e(\Delta_2) = e(\Delta_3) = 1 < e(\Delta_4) = \rho^*(\mathsf{LW4}) = f(\mathsf{LW4}) = e(\mathsf{LW4}) = fhtw(\mathsf{LW4}) = 4/3$. The same plan and size bounds would be obtained by considering $\mathcal{D}$-trees and the same topological order of the variables in Δ_4. ☐

6 Joen: Worst-Case Optimal Multiway Join Algorithm

In this section we introduce Joen, an efficient algorithm for executing a multiway join of all occurrences of a variable or, equivalently, for packing variable orders on a given variable. This is the computational unit of the query plans defined in Section 4. We focus on $\mathcal{E}/\mathcal{D}$-representation systems that support output-bounded query plans and show that for both representation systems Joen takes time linear in its input and output. We first discuss the case of $\mathcal{E}$-representations and then extend the discussion to $\mathcal{D}$-representations.

6.1 Joen$_\mathcal{E}$: Joen on $\mathcal{E}$-representations

Figure 7 depicts the Joen$_\mathcal{E}$ algorithm for computing the join on a variable A. It takes as input a factorization E over an $\mathcal{E}$-tree Δ_{in}, an assignment order ω of the variable occurrences $(A^{(j)})_{j\in[s]}$ of A, which is the order in which we encounter their value assignments as we traverse E top down, and an accumulator μ for these assignments. Its output is a factorization over an $\mathcal{E}$-tree Δ_{out} that is a packing of Δ_{in} on A.

Joen$_\mathcal{E}$ is defined by induction on the structure of E. Consider first that E is a union. Special treatment is necessary in case the variable of E, say $A^{(j)}$, is in ω (according to Δ_{in}), otherwise we return the union of results of Joen$_\mathcal{E}$ on each union term of E. If the variable is in ω, then we record the mapping $\mu[A^{(j)}] = E$. If $A^{(j)}$ is last in ω, i.e., $j = s$, this means that μ has assignments for every variable occurrence $(A^{(j)})_{j\in[s]}$ of A and we can add to the output the intersection of these assignments. If $A^{(j)}$ is in ω but not last, then we

Joen$_{\mathcal{E}}$ ($\mathcal{E}$-representation E, assignment order $\omega = (A^{(j)})_{j\in[s]}$, mappings $\mu : \omega \to \mathcal{E}$)
switch E:
$\bigcup_{l\in[k]} E_l$: **if** $var(E) \in \omega$ **then** {
$\mu[var(E)] = E$
if $var(E) = last(\omega)$ **then return intersect**(ω, μ) **else return** $\langle\rangle$
}
return $\bigcup_{l\in[k]}$ **Joen**$_{\mathcal{E}}(E_l, \omega, \mu)$
$\prod_{l\in[k]} E_l$: **foreach** $l \in [k]$ **do** {
if $schema(E_l) \cap \omega \neq \emptyset$ **then** $newE_l =$ **Joen**$_{\mathcal{E}}(E_l, \omega, \mu)$ **else** $newE_l = E_l$
if $newE_l = \emptyset$ **then return** $\emptyset$
}
return $\prod_{l\in[k]} newE_j$

intersect (assignment order $\omega = (A^{(j)})_{j\in[s]}$, mappings $\mu : \omega \to \mathcal{E}$)
Assume notation: $\forall j \in [s], \exists n_j$ such that $\mu[A^{(j)}] = \bigcup_{l_j\in[n_j]} a^{(j)}_{l_j} \times E^{(j)}_{l_j}$
$result = \emptyset$
foreach $a \in \bigcap_{j\in[s]}[a^{(j)}_1, \ldots, a^{(j)}_{n_j}]$ **do** {
foreach $j \in [s]$ **do let** $l_j \in [n_j]$ be such that $a = a^{(j)}_{l_j}$
$result = result \cup a \times \prod_{j\in[s]} {}^{\uparrow}E^{(j)}_{l_j}$
}
return $result$

Figure 7 Joen$_{\mathcal{E}}$ computes a multiway join on a given variable with occurrences $(A^{(j)})_{j\in[s]}$ and assignment order ω ordered following the traversal of the input $\mathcal{E}$-representation E. While traversing E, μ collects assignments of $(A^{(j)})_{j\in[s]}$ to fragments of E rooted at unions. Once all occurrences get assignments, their intersection is added to the output.

return the nullary relation that acts as identity for the Cartesian product. Now consider that E is a product. We recurse in those product terms of E that may have assignments for $(A^{(j)})_{j\in[s]}$ and keep the other terms untouched (an output $\mathcal{D}$-representation would use references to these terms). If Joen$_{\mathcal{E}}$ returns the empty set for any of these terms, e.g., when the intersection of assignments at a higher recursion depth is empty, then we return the empty set as well since the product of the empty set with anything is the empty set.

The intersection of the assignments for $(A^{(j)})_{j\in[s]}$ can be done efficiently using the unary leapfrog join [16] applied to the unions $(\mu[A^{(j)}])_{j\in[s]}$ represented as ordered arrays. For the purpose of the intersection, we disregard the factorization fragments hanging off the root values in these assignments. Given s such arrays $(L_j)_{j\in[s]}$, where $N_{\min} = \min_{j\in[s]}\{|L_j|\}$ and $N_{\max} = \max_{j\in[s]}\{|L_j|\}$ are the sizes of the smallest and respectively largest array, their intersection takes time $O(N_{\min} \log(N_{\max}/N_{\min}))$, i.e., it takes time linear in the size of the smallest array in our complexity model. From each value a in the intersection, we can now refer back to (instead of copying) all factorization fragments that hang off a in the input E. Using references instead of extensive copies of input factorizations is key to size monotonicity and improved time complexity of $\mathcal{E}$-representations over $\mathcal{F}$-representations.

Joen$_{\mathcal{E}}$ relies on the one-lookahead path property for the variable orders in query plans from Proposition 10(2). We assume without loss of generality that for each Cartesian product in E, the order of its children is compatible with the assignment order ω of occurrences of variable A. The implication of this property is that for the assignment order $\omega = (A^{(j)})_{j\in[s]}$,

Joen$_\mathcal{D}$ ($\mathcal{D}$-representation E, assignment order $\omega = (A^{(j)})_{j\in[s]}$, $keys = key(A)$)
Step 1: Keep definitions and contexts for variables in *keys* $E_1 = \{(\eta_X[\pi_{keys}(ctx)] = U) \mid (\eta_X[ctx] = U) \in E, X \in keys\}$ **Step 2**: Aggregate definitions that have the same context and variable $E_2 = \{(\eta_X[ctx] = \bigcup_{(\eta_X[ctx]=U)\in E_1} U) \mid (\eta_X[ctx] = \_) \in E_1\}$ **Step 3**: Add definitions of A's occurrences $(A^{(j)})_{j\in[s]}$ $E_2 = E_2 \cup \{(\eta_{A^{(j)}}[ctx] = U) \in E \mid j \in [s]\}$ **Step 4**: Compute the multiway join using Joen$_\mathcal{E}$ $E_3 = \textbf{Joen}_\mathcal{E}(E_2, (A^{(j)})_{j\in[s]}, \mu = \emptyset)$ **Step 5**: Remove definitions for A's occurrences $(A^{(j)})_{j\in[s]}$ and add those for A **return** $E \setminus \{(\eta_{A^{(j)}}[ctx] = u) \in E \mid j \in [s]\} \cup \{(\eta_A[ctx] = u) \in E_3\}$

Figure 8 Joen$_\mathcal{D}$ computes a multiway join on a given variable A with occurrences $(A^{(j)})_{j\in[s]}$ and assignment order ω ordered following the traversal of the input $\mathcal{D}$-representation E. It first projects E onto an $\mathcal{E}$-representation E_2 over variables in $key(A)$ and $(A^{(j)})_{j\in[s]}$, then calls Joen$_\mathcal{E}$ on E_2 to compute the definitions for A.

there is a one-to-many relationship between the number of assignments of $A^{(j_1)}$ versus $A^{(j_2)}$ for $1 \le j_1 < j_2 \le s$. This also means that the number of possible total assignments of A's occurrences is at most linear in the size of E. Furthermore, we only need to visit each assignment of ω exactly once and all assignments can be encountered in one pass over E.

▶ **Example 18.** Consider the join on variable C in Section 2 that maps between the $\mathcal{E}$-representations from Figures 3(e) and (f). As we descend below a_0, we find the assignment C_{a_0} for C_2. We eventually reach the assignment C_{b_0} for C_3, which is the last occurrence of C_3 in the assignment order. The intersection of the arrays C_{a_0} and C_{b_0} takes time linear in their sizes since they are equal. We place their intersection under b_0 in the output and discard them. The next assignment for C_3 is C_{b_1}. We trigger a new intersection, now between C_{a_0} and C_{b_1}. This takes constant time, since the latter array has only one value c_0. We continue until we exhaust all assignments of C_3 under a_0 and then move to a_1, etc. ☐

6.2 Joen$_\mathcal{D}$: Joen on $\mathcal{D}$-representations

Figure 8 depicts the Joen$_\mathcal{D}$ algorithm for computing the join on a variable A with variable occurrences $(A^{(j)})_{j\in[s]}$. Its input is a $\mathcal{D}$-representation E presented as a dictionary of definitions of the form $\eta_X[ctx] = U$, which state that the variable X is mapped to a union (sorted array) U of values in the *context ctx*. The context is a tuple of values for all variables in $key(X)$, i.e., the possible values of X in E are uniquely determined by the tuple of values of variables that are ancestors in Δ and on which X depends. A dictionary of definitions is an alternative, more textual presentation of a graphical factorized representation. We can translate in linear time between the two (modulo log factors).

The main challenge of Joen$_\mathcal{D}$, in addition to Joen$_\mathcal{E}$'s, is to avoid performing unnecessary intersections with an assignment for an occurrence of A. To see how this problem may arise, consider that the key of $A^{(j_1)}$ does not include all ancestors in the input $\mathcal{D}$-tree Δ. Then, the same union of values for $A^{(j_1)}$ is reachable in E via all possible tuples of values for excluded ancestors, yet we might only need one intersection with that assignment. This can be the case if a second occurrence $A^{(j_2)}$ of A is root in a separate branch in Δ, so we would only need to intersect the assignments of the two variable occurrences once for every distinct tuple of values of variables $key(A^{(j_1)})$; we give a concrete example in Appendix E.2.

If Joen$_\mathcal{D}$ is a step in a query plan, then the key of A is as in the last variable order of the plan, cf. Section 4. This is a superset of the keys of A's occurrences: $key(A) \supseteq \bigcup_{j\in[s]} key_\Delta(A^{(j)})$. Let $key(A) = \{K_1, \ldots, K_p)\}$. Due to the one-lookahead property (cf. Proposition 10), K_1 to K_p lie along the same path in Δ and $(A^{(j)})_{j\in[s]}$ hang off that path. Without loss of generality, assume their top-down order is K_1 to K_p.

Given the input $\mathcal{D}$-representation E, we construct the functions $F_{K_1}, \ldots, F_{K_p}$ representing the projection of E onto a $\mathcal{T}$-path over $\{K_1, \ldots, K_p\}$ as follows. The dictionary entry η_{K_l} maps each combination of values of variables in $key(K_l)$ to a union of values for K_l. For F_{K_l}, we (i) project away from every combination of values in $key(K_l)$ all values that are not for variables $K_1, \ldots, K_{l-1}$, and (ii) merge the definitions that now have the same context. For instance, given the definitions $\eta_{K_2}[b_1, c_1] = L_1$ and $\eta_{K_2}[b_2, c_1] = L_2$ and assuming we project away the values for B but keep the values for C, we obtain $F_{K_2}[c_1] = L_1 \cup L_2$.

We now construct an $\mathcal{E}$-representation E_2 as follows. We first construct a factorization E_1 over the $\mathcal{T}$-path from K_1 to K_p. It becomes an $\mathcal{E}$-tree Δ_e and satisfying the one-lookahead property (cf. Proposition 10) once we add the definitions for $(A^{(j)})_{j\in[s]}$ having values for some of K_1 to K_p as context.

We next run Joen$_\mathcal{E}$ on E_2 to compute the multiway join on $(A^{(j)})_{j\in[s]}$ and obtain an $\mathcal{F}$-representation E_3 over the $\mathcal{T}$-path from K_1 over K_p to A. We collect from E_3 the definitions η_A for variable A by taking each possible tuple of values for $K_1, \ldots, K_p$, as the context of a definition for a union of values for A. We construct the output $\mathcal{D}$-representation as E, where we remove all definitions $\eta_{A^{(j)}}$ for occurrences of A and add instead the new definitions η_A. An optional final step is to clean empty definitions arising from empty intersections (not in the pseudocode, same as in Joen$_\mathcal{E}$).

6.3 Summing Up

We can now state our main result on query plans with Joen.

▶ **Theorem 19.** *Given a step in a join query plan, where the input is a $\mathcal{D}$-representation IN over $\mathcal{D}$-tree Δ and A is the variable packed at Δ, Joen$_\mathcal{D}$ computes a $\mathcal{D}$-representation OUT of the join result over a $\mathcal{D}$-tree that is the packing of Δ on A in time $O(|IN| + |OUT|)$.*

Theorem 19 readily applies to less succinct specializations of $\mathcal{D}$-representations such as $\mathcal{E}$-representations, $\mathcal{F}$-representations, and $\mathcal{T}$-representations.

To state our main result, we recall the tight bounds on the sizes of factorized representations of join results: Given a join query Q, for any database $\mathbf{D}$, the join result $Q(\mathbf{D})$ admits: a $\mathcal{T}$-representation of size $\Theta(|\mathbf{D}|^{\rho^*(Q)})$ [2]; an $\mathcal{F}$-representation of size $\Theta(|\mathbf{D}|^{f(Q)})$ [12]; an $\mathcal{E}$-representation of size $\Theta(|\mathbf{D}|^{e(Q)})$; and a $\mathcal{D}$-representation of size $\Theta(|\mathbf{D}|^{fhtw(Q)})$ [13].

An immediate corollary of the previous result, of the time complexity of Joen, and of the monotonically width-increasing property of query plans for $\mathcal{E}/\mathcal{D}$-representations is that the $\mathcal{E}/\mathcal{D}$-representation systems support worst-case optimal join-at-a-time query processing.

▶ **Corollary 20** (Th. 19, 15). *Given a join query Q and any $\mathcal{T}$-representation $\mathbf{D}$.*

There is a query plan with $\mathcal{D}$-representations as intermediate results that computes a $\mathcal{D}$-representation of the query result in time $O(|\mathbf{D}|^{fhtw(Q)})$.

There is a query plan with $\mathcal{E}/\mathcal{D}$-representations as intermediate results that computes the query result: as an $\mathcal{E}$-representation in time $O(|\mathbf{D}|^{e(Q)})$; as an $\mathcal{F}$-representation in time $O(|\mathbf{D}|^{f(Q)})$; and as a $\mathcal{T}$-representation in time $O(|\mathbf{D}|^{\rho^*(Q)})$.

7 Related Work

We classify the join algorithms into join, query, or relation at a time.

Our approach falls into the first category. It builds on prior work on: (1) tight size bounds for $\mathcal{T}$-representations of results to join queries [2] and for $\mathcal{F}/\mathcal{D}$-representations of results to conjunctive queries [13]; and (2) the worst-case optimal query-at-a-time join algorithms NPRR [10] and LeapFrog TrieJoin (LFTJ) [16] for $\mathcal{T}$-representations of join results and FDB [13] for $\mathcal{F}/\mathcal{D}$-representations of join results. FDB defaults to LFTJ for $\mathcal{T}$-representations. Our query algorithm is a modular, join-at-a-time version of FDB. Whereas FDB explores the space of variable mappings depth-first, our approach explores this space breadth-first. That is, given a variable order, FDB searches a value mapping for the first variable and then for the second variable and so on. When all variables have mappings, FDB backtracks. In contrast, our approach first computes the possible mappings of the first variable, then those of the second variable and so on.

Standard query plans with joins at inner nodes and relations at leaves are prime examples in the relation-at-a-time category. In this setting, Yannakakis algorithm [17] for acyclic queries has been recently adapted to produce worst-case optimal query plans [7]. It remains an open question whether the result for acyclic queries can be adapted to the representation systems discussed in this paper. For cyclic queries, relation-at-a-time worst-case optimality is not possible for any of the considered four representation systems, cf. the triangle query for $\mathcal{T}/\mathcal{F}$-representations, and LW4 query for $\mathcal{E}/\mathcal{D}$-representations. If we first join three of the four relations in the Loomis-Whitney query of length four from Figure 6, we obtain intermediate results whose fractional edge cover number (and indeed, the other wdiths discussed in this paper) equals $3/2$, whereas the fractional edge cover number (and the other widths) for the query result is $4/3$.

In a distributed setting, there are relation-at-a-time [9] and query-at-a-time [5] join processing approaches with worst-case optimal communication cost. State-of-the-art monolithic approaches shuffle the data across the servers and run LFTJ or classical (suboptimal) query engines locally at each server.

8 Conclusion

Our work studies worst-case optimal join-at-a-time processing across four factorized representation systems. The three key aspects of our approach, namely factorized representations, worst-case optimality, and join-at-a-time processing, are useful in a variety of settings.

The asymptotic size gap between the various factorized representations, as depicted in Figure 1, translates in practice to orders-of-magnitude performance improvement for join processing [4] and subsequent aggregates [3] and machine learning [14].

The recent FAQ generalization of FDB to the Boolean and sum-product semirings [8] can be immediately applied to our work as well. The FAQ framework captures frequently asked questions across Computer Science, including counting (quantified) conjunctive queries, inference in probabilistic graphical models, matrix multiplication, and constraint satisfaction.

By enabling join-at-a-time computation, we increase the modularity of worst-case optimal join algorithms and narrow the gap between the theory of worst-case optimal join algorithms and the standard commercial relational engines that use query plans with intermediate results. Furthermore, modularity is prerequisite for distributed join-at-a-time query processing, which is the norm in commercial systems [15]. Joen may be useful to develop novel policies for distributed query processing [1] having worst-case optimality guarantees.

Acknowledgements. The authors are indebted to Hung Ngo and Dan Suciu for discussions on the topic of this paper and to Joe Kirk for his implementation of Joen$_{\mathcal{E}}$.

References

1 Tom J. Ameloot, Gaetano Geck, Bas Ketsman, Frank Neven, and Thomas Schwentick. Parallel-correctness and transferability for conjunctive queries. In *PODS*, pages 47–58, 2015.

2 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. In *FOCS*, pages 739–748, 2008.

3 Nurzhan Bakibayev, Tomás Kociský, Dan Olteanu, and Jakub Závodný. Aggregation and ordering in factorised databases. *PVLDB*, 6(14):1990–2001, 2013.

4 Nurzhan Bakibayev, Dan Olteanu, and Jakub Závodný. FDB: A query engine for factorised relational databases. *PVLDB*, 5(11):1232–1243, 2012.

5 Paul Beame, Paraschos Koutris, and Dan Suciu. Skew in parallel query processing. In *PODS*, pages 212–223, 2014.

6 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1):4:1–4:20, 2014.

7 Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. Joins via geometric resolutions: Worst-case and beyond. In *PODS*, pages 213–228, 2015.

8 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: Questions asked frequently. *CoRR*, abs/1504.04044, 2015.

9 Paraschos Koutris, Paul Beame, and Dan Suciu. Worst-case optimal algorithms for parallel query processing. In *ICDT*, pages 8:1–8:18, 2016.

10 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. In *PODS*, pages 37–48, 2012.

11 Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: New developments in the theory of join algorithms. *SIGMOD Record*, 42(4):5–16, 2013.

12 Dan Olteanu and Jakub Závodný. Factorised representations of query results: size bounds and readability. In *ICDT*, pages 285–298, 2012.

13 Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Trans. Database Syst.*, 40(1):2, 2015.

14 Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *SIGMOD*, 2016.

15 Jeff Shute, Radek Vingralek, Bart Samwel, Ben Handy, Chad Whipkey, Eric Rollins, Mircea Oancea, Kyle Littlefield, David Menestrina, Stephan Ellner, John Cieslewicz, Ian Rae, Traian Stancescu, and Himani Apte. F1: A distributed SQL database that scales. *PVLDB*, 6(11):1068–1079, 2013.

16 Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *ICDT*, pages 96–106, 2014.

17 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.

A Additional Material for Section 2

Worst-case Optimality for Triangle Query

We generalize the example in the introduction to arbitrary relations R_1, R_2, and R_3 and show that Joen needs time $O(\sqrt{|R_1| \cdot |R_2| \cdot |R_3|})$, which is worst-case optimal [2]. Our proof is inspired by techniques introduced for showing worst-case optimality for a generalization of NPRR and LFTJ [11]. We show this for a linear query plan that executes the three join conditions on A, B, and C in sequence; it can be shown similarly for any other possible linear plan that is a permutation of this one.

The result of Joen on A is the $\mathcal{E}$-representation J_1, the result of Joen on B is the $\mathcal{E}$-representation J_2, and the final result J_3 is a $\mathcal{T}$-representation obtained by executing Joen on C. The variable orders of the three factorized representations are as shown in Figure 3(d-f).

To compute J_1, Joen intersects the ordered lists of A-values in the $\mathcal{T}$-representations of R_1 and R_2 in time

$$\min(|\pi_{A_1}(R_1)|, |\pi_{A_2}(R_2)|) \leq \sqrt{|\pi_{A_1}(R_1)| \cdot |\pi_{A_2}(R_2)|} \leq \sqrt{|R_1| \cdot |R_2|}.$$

The first inequality above uses the inequality $\min(x, y) \leq \sqrt{x \cdot y}$ for $x, y \geq 0$. Each of the A-values in the intersection inherits the pointers to their unions of B-values and C-values from the input $\mathcal{T}$-representations. This saves time linear in the sizes of R_1 and R_2 by avoiding to copy these unions from the input to J_1; even in case we would copy these unions, the overall time stays below the worst-case optimal time for the entire triangle query. The structure of J_1 is given by the $\mathcal{E}$-tree in Figure 3(d), where the dotted edges mean that we use references to connect A-values to their corresponding B and C-values.

For the join condition on B, we follow the branch of each A-value in J_1 and intersect the B_1-values in its union with the list of B_2-values in S. Let $L_A = \pi_{A_1}(R_1) \cap \pi_{A_2}(R_2)$ be the list of A-values in J_1. The number of steps to compute J_2 is then[1]:

$$\sum_{a \in L_A} \min(|\pi_{B_1}\sigma_{A_1=a}(R_1)|, |\pi_{B_3}(R_3)|) \leq \sum_{a \in L_A} \sqrt{|\pi_{B_1}\sigma_{A_1=a}(R_1)| \cdot |\pi_{B_3}(R_3)|} =$$
$$\sqrt{|\pi_{B_3}(R_3)|} \cdot \sum_{a \in L_A} \sqrt{|\pi_{B_1}\sigma_{A_1=a}(R_1)|} \leq \sqrt{|\pi_{B_3}(R_3)|} \cdot \sqrt{\sum_{a \in L_A} |\pi_{B_1}\sigma_{A_1=a}(R_1)|} \cdot \sqrt{\sum_{a \in L_A} 1}$$
$$\leq \sqrt{|\pi_{B_3}(R_3)|} \cdot \sqrt{|\pi_{B_1}(R_1)|} \cdot \sqrt{|\pi_{A_2}(R_2)|} \leq \sqrt{|R_1| \cdot |R_2| \cdot |R_3|}.$$

For the second inequality, we use the Cauchy-Schwarz inequality:

$$\Big(\sum_{a \in L_A} x_a \cdot y_a\Big)^2 \leq \Big(\sum_{a \in L_A} x_a^2\Big) \cdot \Big(\sum_{a \in L_A} y_a^2\Big), \text{ where } y_a = 1 \text{ in our case.}$$

To compute J_2, we do not need to copy the unions of C_3-values from the $\mathcal{T}$-representation of R_3 for each matching B-value. Instead, we have references from J_2 to these unions in R_3 similarly to the unions of C_2-values in R_2; the $\mathcal{E}$-tree of J_2 is given in Figure 3(e). Using references instead of copying the unions is necessary for worst-case optimality: If we would copy in J_2 the unions of C-values from the $\mathcal{T}$-representations of R_3 and R_2, then the size of J_2 would be quadratic in the input size! This can be already seen for the introductory example: If we would copy C_{b_0} under each A-value in J_2, then the size of J_2 would become

[1] An alternative upper bound (that is smaller) is:
$\sum_{a \in L_A} \min(|\pi_{B_1}\sigma_{A_1=a}(R_1)|, |\pi_{B_3}(R_3)|) \leq \sum_{a \in L_A} |\pi_{B_1}\sigma_{A_1=a}(R_1)| \leq |R_1|$.

at least $(m+1)^2$ while each input relation is of size linear in m, since each value a_i would have all of c_j values underneath.

Finally, we compute J_3 by computing the join on C: Under each A-value a, we intersect the union of C_2-values with each union of C_3-values under each B-value b under a. Let L_B^a be the list of B-values under the A-value a in J_2. The number of computation steps is then:

$$\sum_{a\in L_A}\sum_{b\in L_B^a}\min(|\pi_C\sigma_{B_3=b}(R_3)|, |\pi_{C_2}\sigma_{A_2=a}(R_2)|).$$

The inner sum can be expanded similarly to the case for J_2:

$$\begin{aligned}
&\sum_{b\in L_B^a}\min(|\pi_{C_3}\sigma_{B_3=b}(R_3)|, |\pi_{C_2}\sigma_{A_2=a}(R_2)|) \le \sum_{b\in L_B^a}\sqrt{|\pi_{C_3}\sigma_{B_3=b}(R_3)|\cdot|\pi_{C_2}\sigma_{A_2=a}(R_2)|}\\
&=\sqrt{|\pi_{C_2}\sigma_{A_2=a}(R_2)|}\cdot\sum_{b\in L_B^a}\sqrt{|\pi_{C_3}\sigma_{B_3=b}(R_3)|}\\
&\le\sqrt{|\pi_{C_2}\sigma_{A_2=a}(R_2)|}\cdot\sqrt{\sum_{b\in L_B^a}|\pi_{C_3}\sigma_{B_3=b}(R_3)|}\cdot\sqrt{\sum_{b\in L_B^a}1}\\
&\le\sqrt{|\pi_{C_2}\sigma_{A_2=a}(R_2)|}\cdot\sqrt{|\pi_{C_3}(R_3)|}\cdot\sqrt{|\pi_{B_1}\sigma_{A_1=a}(R_1)|}.
\end{aligned}$$

We can now plug this into the first sum and obtain:

$$\begin{aligned}
&\sum_{a\in L_A}(\sqrt{|\pi_{C_2}\sigma_{A_2=a}(R_2)|}\cdot\sqrt{|\pi_{C_3}(R_3)|}\cdot\sqrt{|\pi_{B_1}\sigma_{A_1=a}(R_1)|})\\
=&\sqrt{|\pi_{C_3}(R_3)|}\cdot\sum_{a\in L_A}(\sqrt{|\pi_{C_2}\sigma_{A_2=a}(R_2)|}\cdot\sqrt{|\pi_{B_1}\sigma_{A_1=a}(R_1)|})\\
\le&\sqrt{|\pi_{C_3}(R_3)|}\cdot\sqrt{\sum_{a\in L_A}|\pi_{C_2}\sigma_{A_2=a}(R_2)|}\cdot\sqrt{\sum_{a\in L_A}|\pi_{B_1}\sigma_{A_1=a}(R_1)|}\\
\le&\sqrt{|\pi_{C_3}(R_3)|}\cdot\sqrt{|\pi_{C_2}(R_2)|}\cdot\sqrt{|\pi_{B_1}(R_1)|}\le\sqrt{|R_3|\cdot|R_2|\cdot|R_1|}.
\end{aligned}$$

This shows that Joen can compute the triangle query worst-case optimally and one join condition at a time. To recall, this is not the case for the relational query plans since the first join can already lead to a quadratic time complexity! The key ingredient exploited by Joen to achieve worst-case optimality is the factorized representation of the intermediate join results, which ensures that the join J_1 on A has size linear in the input size; the $\mathcal{E}$-representation using references to previously computed representations ensures that the subsequent join on B stays within the required optimal bounds.

B Additional Material for Section 3

B.1 Preliminaries

Databases. A *schema* Σ is a set of attributes. We consider databases $\mathbf{D}$ of n relations $R_1,\ldots,R_n$, whose schemas have attributes denoted by capital letters. Attribute values are denoted by lowercase letters. We assume that all attribute domains are totally ordered (we order a domain arbitrarily if there is no natural total order). We denote by the size $|R_i|$ of relation R_i the number of tuples in R_i. Let $N=\max_{i\in[n]}(|R_i|)$. The size $|\mathbf{D}|$ of a database $\mathbf{D}$ is the sum of the sizes of its relations.

Queries. We consider equi-join queries, e.g., the triangle query $Q_\triangleleft$ in Section 2. When referring to distinct occurrences of a variable in a query, we index them using the index of their relation symbol. For instance, the occurrence of variable A is denoted by A_1 in relation symbol R_1 and by A_2 in R_2 (this notation disallows trivial join conditions on variables within the same relation symbol). For a variable A, $rel(A)$ denotes the set of relation symbols that have occurrences of A. The size $|Q|$ of a query Q is the number n of its relations.

▶ **Definition 21.** Given a join query Q over relations $R_1, \ldots, R_n$ and a set of variables X, the X*-restriction* of Q is the join query Q_X that is Q where all variables not in X are removed and where each relation R_i is replaced by $R_i^X = \pi_X(R_i)$ for $i \in [n]$.

For example, the $\{A, B\}$-restriction of the triangle query $R_1(A, B)$, $R_2(A, C), R_3(B, C)$ is $R_1^X(A, B), R_2^X(A), R_3^X(B)$.

Complexity assumptions. We ignore factors logarithmic in the size of the database. We consider data complexity, where the query is fixed, and measure the complexity as a function of the database size and ignore factors depending on the query size.

Notation. Given a natural number m, by $[m]$ we denote the set $\{1, \ldots, m\}$.

B.2 Size Measures and Relative Succinctness

For a join query Q and a variable order Δ for Q, the factorized result of Q over Δ is unique up to commutativity of product and union. There may be however several possible variable orders for Q and they define factorized representations of different sizes. We next review size measures for factorized representations in our four representation systems. They are defined on the query hypergraph: For Q, the hypergraph $H(Q) = (V, E)$ has one node in the set V per query variable in Q and one hyperedge in the set E per relation in Q. Figure 9(a) depicts the hypergraph of the triangle query.

An edge cover is a subset of (hyper)edges of $H(Q)$ such that each node appears in at least one edge. Edge cover can be formulated as an integer programming problem by assigning to each edge R_i a weight x_i that can be 1 if R_i is part of the cover and 0 otherwise. The size of an edge cover upper bounds the size of the query result, since the Cartesian product of the relations in the cover includes the query result:

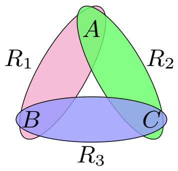
$$|Q(\mathbf{D})| \leq |R_1|^{x_1} \cdot \ldots \cdot |R_n|^{x_n} \leq N^{\sum_{i=1}^{n} x_i}.$$

By minimizing the size of the edge cover, we can obtain a more accurate upper bound on the size of the query result. Atserias, Grohe, and Marx (henceforth AGM) showed that this bound becomes tight for fractional weights [2]. Minimizing the sum of the weights thus becomes the objective of a linear program instead of an integer program.

▶ **Definition 22** ([2])**.** Given a join query Q over a database $\mathbf{D} = (R_1, \ldots, R_n)$, the *fractional edge cover number* $\rho^*(Q)$ is the cost of an optimal solution to the linear program with variables $\{x_i\}_{i=1}^n$:

$$\text{minimize} \quad \textstyle\sum_{i=1}^n x_i \qquad \text{subject to} \sum_{R_i \in rel(A)} x_i \geq 1 \quad \text{for each query variable } A$$
$$x_i \geq 0 \qquad \text{for each } 1 \leq i \leq n. \qquad \square$$

Figure 9(b) gives the linear program for the fractional edge cover of the triangle query $Q_\triangleleft$. An optimal solution is $\rho^*(Q_\triangleleft) = 3/2$ with $x_1 = x_2 = x_3 = 1/2$. Consequently, the result of the triangle query has $O(N^{3/2})$ tuples (recall that by N we denote the size of the largest relation). In this paper, for ease of presentation of our results, we assume that all relations



(a) Hypergraph.

$$
\begin{array}{lccccc}
\multicolumn{6}{l}{\textit{minimize } x_1 + x_2 + x_3} \\
\multicolumn{6}{l}{\textit{subject to}} \\
A: & x_1 & + & x_2 & & \geq 1 \\
B: & x_1 & & & + \; x_3 & \geq 1 \\
C: & & & x_2 & + \; x_3 & \geq 1
\end{array}
$$

(b) Linear program.

Figure 9 Hypergraph for the triangle query $Q_{\triangleleft}$ and the inequalities for query variables in the linear program for computing the tight size bound on the query result.

are of the same size N. At the end of the section, we discuss how our setting naturally adapts to the general case of relations with arbitrary sizes. Moreover, the AGM bound is tight. For instance, for the triangle query $Q_{\triangleleft}$, there exist classes of databases for which the result size is at least $\Omega(N^{3/2})$.

The *factorization width*, denoted by $f(Q)$, is the fractional edge cover number of a subquery of Q [13]. For an $\mathcal{F}$-representation over an $\mathcal{F}$-tree Δ of a join query Q, the number of values of a variable A, denoted s_A, is dependent on the number of possible tuples of values of its ancestors, whose set is $key(A)$, and is independent of the number of values for variables that are not on the same branch. A tight bound on s_A is then given by the fractional edge cover number of the join query that is a $(key(A) \cup \{A\})$-restriction of Q. Then, an upper bound on the size of the $\mathcal{F}$-representation over a specific Δ is the maximum over all variables in Δ of the number of values of A:

$$f(\Delta) = \max\{\rho^*(Q_{key(A)\cup\{A\}})|A \text{ is variable in } \Delta\}$$

The factorization width $f(Q)$ is then the minimum over all possible $\mathcal{F}$-trees of the previous upper bound:

$$f(Q) = \min\{f(\Delta)|\Delta \text{ is an } \mathcal{F}\text{-tree of } Q\}$$

The *e-width* $e(Q)$ and the *fractional hypertree width* $\mathit{fhtw}(Q)$ [13] are defined similarly to the factorization width $f(Q)$, with the difference that the key of a variable may not be the set of all ancestors as for $\mathcal{F}$-trees. In other words, we iterate over $\mathcal{E}$-trees and respectively $\mathcal{D}$-trees instead of only over their strict subset of $\mathcal{F}$-trees:

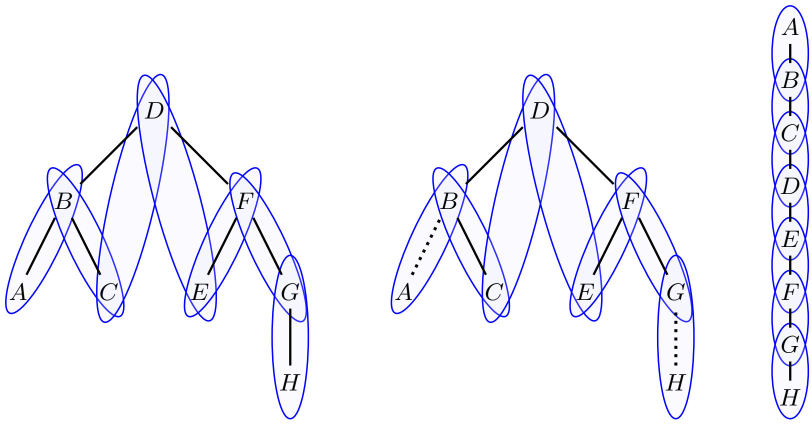
$$e(Q) = \min\{f(\Delta)|\Delta \text{ is an } \mathcal{E}\text{-tree of } Q\}, \qquad \mathit{fhtw}(Q) = \min\{f(\Delta)|\Delta \text{ is a } \mathcal{D}\text{-tree of } Q\}.$$

We present the relationships between the representation systems and their widths in Figure 1. There are queries for which $\rho^*(Q) = |Q|$ while $f(Q) = 1$, e.g., hierarchical queries Q where each relation symbol has a variable not appearing in join conditions. For path queries Q, $\mathit{fhtw}(Q) = 1$ while $f(Q) = \log\lceil|Q|\rceil$ [13]. The relation between $e(Q)$ and $f(Q)$ is new:

▶ **Proposition 23.** *Given a join query Q, it holds that $f(Q) \geq e(Q) \geq f(Q) - 1$.*

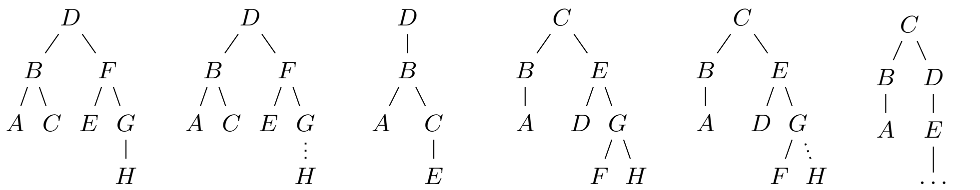
The above widths give asymptotically tight bounds on factorization sizes in the four representation systems (recall that we assume data complexity and all relations of size N).

▶ **Proposition 24.** *Given a join query Q, for any database $\mathbf{D}$ of size N, the join result $Q(\mathbf{D})$ admits: a $\mathcal{T}$-representation of size $\Theta(N^{\rho^*(Q)})$ [2]; an $\mathcal{F}$-representation of size $\Theta(N^{f(Q)})$ [12]; an $\mathcal{E}$-representation of size $\Theta(N^{e(Q)})$; and a $\mathcal{D}$-representation of size $\Theta(N^{\mathit{fhtw}(Q)})$ [13].*



	(a) $\mathcal{F}$-tree	**(b)** $\mathcal{E}$-tree	**(c)** $\mathcal{D}$-tree
$key(A)$	$\{B, D\}$	$\{B\}$	$\emptyset$
$key(B)$	$\{D\}$	$\{D\}$	$\{A\}$
$key(C)$	$\{B, D\}$	$\{B, D\}$	$\{B\}$
$key(D)$	$\emptyset$	$\emptyset$	$\{C\}$
$key(E)$	$\{D, F\}$	$\{D, F\}$	$\{D\}$
$key(F)$	$\{D\}$	$\{D\}$	$\{E\}$
$key(G)$	$\{D, F\}$	$\{D, F\}$	$\{F\}$
$key(H)$	$\{D, F, G\}$	$\{G\}$	$\{G\}$

Figure 10 Variable orders $\mathcal{F}$-tree, $\mathcal{E}$-tree, and $\mathcal{D}$-tree for query Path7 in Section B.3.



(a) $\mathcal{F}$-tree. **(b)** $\mathcal{E}$-tree. **(c)** $\mathcal{D}$-tree. **(d)** $\mathcal{F}$-tree. **(e)** $\mathcal{E}$-tree. **(f)** $\mathcal{D}$-tree.

Figure 11 Example of variable orders for queries Q_2 and Q_3 in Section B.3.

Our definitions of widths assume for simplicity that all relations have the same size N. Our results carry over to definitions that take individual relation sizes into account. All we need is change Definition 22 to include relation sizes in the objective of the linear program [2]) as follows. Take a join query Q over a database $\mathbf{D} = (R_1, \ldots, R_n)$ and a fractional edge cover $(x_1, \ldots, x_n)$. The fractional edge cover number becomes

$$\rho^*(Q) = \sum_{i=1}^{n} x_i \log |R_i|.$$

Our definitions for $f(\Delta)$, $f(Q)$, $e(Q)$, and $fhtw(Q)$ remain the same but they rely now on this revisited notion of $\rho^*(Q)$. Then, in Proposition 24, for every representation system $\mathcal{X} \in \{\mathcal{T}, \mathcal{F}, \mathcal{E}, \mathcal{D}\}$ and corresponding width measure $w_{\mathcal{X}} \in \{\rho^*, f, e, fhtw\}$, the join result $Q(\mathbf{D})$ admits an $\mathcal{X}$-representation of size $\Theta(2^{w_{\mathcal{X}}(Q)})$.

B.3 Examples of Width Measures for Acyclic and Cyclic Queries

We illustrate the size measures for the following Path7 query:

$$Q = R_1(A, B), R_2(B, C), R_3(C, D), R_4(D, E), R_5(E, F), R_6(F, G), R_7(G, H).$$

For $\mathcal{T}$-representations, $\rho^*(Q) = 4$ where we set $x_1 = x_3 = x_5 = x_7 = 1$ to satisfy:

$$A : x_1 \geq 1, \qquad B : x_1 + x_2 \geq 1, \qquad C : x_2 + x_3 \geq 1, \qquad D : x_3 + x_4 \geq 1,$$
$$E : x_4 + x_5 \geq 1, \qquad F : x_5 + x_6 \geq 1, \qquad G : x_6 + x_7 \geq 1, \qquad H : x_7 \geq 1.$$

For $\mathcal{F}$-representations, $f(Q) = 3$. Figure 10(a) gives an optimal $\mathcal{F}$-tree that matches this parameter, where:

$$\rho^*(Q_{\{A,B,D\}}) = 2, \qquad \rho^*(Q_{\{C,B,D\}}) = 2, \qquad \rho^*(Q_{\{E,F,D\}}) = 2, \qquad \rho^*(Q_{\{H,G,F,D\}}) = 3.$$

Since for $\mathcal{F}$-trees each variable has all ancestors in its key, it suffices to only look at its leaves to compute $f(Q)$. For $\mathcal{E}$-representations, $e(Q) = 2$ as witnessed by the $\mathcal{E}$-tree in Figure 10(b). In contrast to the previous $\mathcal{F}$-tree, the keys for A and H are only their parents (their edges are dotted), and $\rho^*(Q_{\{H\}\cup key(H)}) = 1$ as opposed to 3 for the previous $\mathcal{F}$-tree. For $\mathcal{D}$-representations, *fhtw*$(Q) = 1$ since Q is acyclic. A $\mathcal{D}$-tree would be the path from A to H, cf. Figure 10(c).

The previous query is acyclic. We next discuss the width measures for two cyclic queries.

- **We show $f(Q_2) > e(Q_2)$ for the cyclic query $Q_2 = Q_1, R_8(A, D), R_9(B, D)$.**

For $\mathcal{T}$-representations of Q_2's result, $\rho^*(Q_2) = 4$. This is obtained using the same variables assignments as for Q_1 in the program:

$$A : x_1 + x_8 \geq 1, \qquad B : x_1 + x_2 + x_9 \geq 1,$$
$$C : x_2 + x_3 \geq 1, \qquad D : x_3 + x_4 + x_8 + x_9 \geq 1,$$
$$E : x_4 + x_5 \geq 1, \qquad F : x_5 + x_6 \geq 1,$$
$$G : x_6 + x_7 \geq 1, \qquad H : x_7 \geq 1.$$

For $\mathcal{F}$-representations of Q_2's result, $f(Q_2) = 3$. This is obtained using the $\mathcal{F}$-tree in Figure 11(a). The path ending in A has the f-width $3/2$ (the f-width of the query defining the leaf values), which does not affect the maximum f-width that is attained for the path ending in H.

For $\mathcal{E}$-representations of Q_2's result, $e(Q_2) = 2$. This is obtained using the $\mathcal{E}$-tree in Figure 11(b); the maximum e-width is attained for the paths ending in C or E.

For $\mathcal{D}$-representations of Q_2's result, *fhtw*$(Q_2) = 2$. This is obtained using the $\mathcal{D}$-tree in Figure 11(c).

- **We show $e(Q_3) >$ *fhtw*(Q_3) for the cyclic query $Q_3 = Q_1, R_{10}(A, C)$.**

For $\mathcal{T}$-representations of Q_3's result, $\rho^*(Q_3) = 4$. This is obtained using the same weight assignments as for Q_1 in the program:

$$A : x_1 + x_{10} \geq 1, \quad B : x_1 + x_2 \geq 1, \quad C : x_2 + x_3 + x_{10} \geq 1, \quad D : x_3 + x_4 \geq 1,$$
$$E : x_4 + x_5 \geq 1, \quad F : x_5 + x_6 \geq 1, \quad G : x_6 + x_7 \geq 1, \quad H : x_7 \geq 1.$$

For $\mathcal{F}$-representations of Q_3's result, $f(Q_3) = 3$. This is obtained for the $\mathcal{F}$-tree in Figure 11(d); the paths ending in F or H has the maximum f-wdith of 3.

For $\mathcal{E}$-representations of Q_3's result, $e(Q_3) = 3$. This is obtained for the $\mathcal{E}$-tree in Figure 11(e); the path ending in F has the maximum e-width of 3.

For $\mathcal{D}$-representations of Q_3's result, *fhtw*$(Q_3) = 3/2$. This is obtained for the $\mathcal{D}$-tree in Figure 11(f); the path ending in A has the maximum fractional hypertree width of $3/2$.

B.4 Proofs

Additional notation. Given a variable order Δ and a variable A, by $anc_\Delta(A)$ we denote the set of variables on the path from the root to A.

Proof of Proposition 23

We prove that given a join query Q, it holds that $f(Q) \geq e(Q) \geq f(Q) - 1$.

Since every $\mathcal{E}$-tree is also an $\mathcal{F}$-tree, $f(Q) \geq e(Q)$ follows immediately from the definitions.

Assume towards a contradiction that there exists a query Q for which $e(Q) < f(Q) - 1$. Take the $\mathcal{E}$-tree Δ that witnesses the e-width i.e., $e(Q) = f(\Delta)$. Transform the $\mathcal{E}$-tree Δ into an $\mathcal{F}$-tree Δ' by setting $key_{\Delta'}(A) = anc_{\Delta'}(A)$ for all variables. Consequently, for each leaf A in Δ', the linear program for computing $\rho^*(Q_{key_{\Delta'}(A)\cup\{A\}})$ contains:

- for each ancestor $B \in anc_{\Delta'}(A)$ such that $key_\Delta(B) = anc_\Delta(B)$, precisely the same inequality as for computing $\rho^*(Q_{key_\Delta(B)\cup\{B\}})$;
- for each ancestor $C \in anc_{\Delta'}(A)$ such that $key_\Delta(C)$ contains only the ancestors of C that occur in the same relation R with C, the inequality $x_R \geq 1$ (we recall that the definition of the $\mathcal{E}$-trees implies the existence of an unique such relation R).

Consequently, $f(\Delta') \leq f(\Delta) + 1$, thus $f(\Delta') \leq e(Q) + 1$, thus $f(\Delta') < f(Q)$. By definition, there does not exist an $\mathcal{F}$-tree having $f(\Delta') < f(Q)$, hence we have a contradiction. In conclusion, $f(Q) \geq e(Q) \geq f(Q) - 1$.

Proof of Proposition 24

We prove that given a join query Q, for every database $\mathbf{D}$ of size N, the join result $Q(\mathbf{D})$ admits an $\mathcal{E}$-representation of size $\Theta(N^{e(Q)})$. Similar results have been already proven for $\mathcal{T}$-representations [2], $\mathcal{F}$-representations [12], and $\mathcal{D}$-representations [13].

The *upper bound* follows directly from the definitions in Section B.2. As for the *lower bound*, if follows from the same definitions and the techniques used in [13] (Section 7.4) to prove the similar result for $\mathcal{D}$-representations. More precisely, their result of interest is that given a fixed query Q, there exist arbitrarily large databases $\mathbf{D}$ such that the number of A-values in the representation of $Q(\mathbf{D})$ is at least $|\mathbf{D}|^{\rho^*(Q_{key(A)\cup\{A\}})}$. Our $\mathcal{E}$-trees are nothing else than a particular case of the $\mathcal{D}$-trees considered there.

C Additional Material for Section 4

Proof of Proposition 10

Throughout this section, we assume a plan $\Delta_0, \ldots, \Delta_m$ over m variables.

Moreover, we assume without loss of generality that we have a virtual root *root* of the variable order such that the actual roots of the variable order become children of this virtual *root*. We may denote by *root* the lowest common ancestor (*lca*) of an actual root of the factorization and any other variable.

Part 1 of Proposition 10. For any variable (occurrence) A it holds that: $\forall i \in [m]$: $key_{\Delta_{i-1}}(A) \subseteq key_{\Delta_i}(A)$.

Take a variable A from Δ_{i-1}.

If A is a not yet packed variable, then its key consists only of its ancestors with whom it occurs in the same relation. If A remains not yet packed in Δ_i, then $key_{\Delta_{i-1}}(A) = key_{\Delta_i}(A)$.

Moreover, if A is packed in Δ_i, the variables with whom it occurs in the same relation are part of the key of A in Δ_i.

Next, we consider the case where A is packed in Δ_{i-1} and assume towards a contradiction that $key_{\Delta_{i-1}}(A) \nsubseteq key_{\Delta_i}(A)$ i.e., there exists a variable $B \in key_{\Delta_{i-1}}(A)$ such that $B \notin key_{\Delta_i}(A)$. There are two cases for B being in $key_{\Delta_{i-1}}(A)$:

(1) A and B occur in a same relation, or

(2) there is no relation containing A and B at the same time, but there exists a child C of A such that $B \in key_{\Delta_{i-1}}(C)$ (cf. Definition 2).

If (1) holds, then $B \in key_{\Delta_i}(A)$ that contradicts the hypothesis. Assuming that (2) holds, we identify two cases for which $B \in key_{\Delta_{i-1}}(C)$ and we now reiterate the reasoning by considering two cases (2.1) and (2.2) (where the inner 1 and 2 are as above), and so on. However, since there is a fixed number of descendants of A in Δ_{i-1}, then after a fixed number of iterations we do not have any other child C to consider case (2) anymore, hence the only case is (1). This implies a contradiction, hence we conclude that $key_{\Delta_{i-1}}(A) \subseteq key_{\Delta_i}(A)$.

Before proving Part 2, we show an auxiliary result.

▶ **Lemma 25.** For all Δ_{i-1} ($i \in [m]$), if A is the variable to pack to obtain Δ_i, then for all pairs of occurrences of A in Δ_{i-1}, it holds that at least one of them is a child of their lowest common ancestor in Δ_{i-1}. ◻

Proof. Take without loss of generality $A^{(1)}$ and $A^{(2)}$ as a pair of occurrences of A in Δ_{i-1}. Suppose towards a contradiction that the lowest common ancestor of $A^{(1)}$ and $A^{(2)}$ in Δ_{i-1} has among its children two different variables B and C such that $A^{(1)}$ and $A^{(2)}$ are their descendants, respectively, and $B \in key_{\Delta_{i-1}}(A^{(1)})$ and $C \in key_{\Delta_{i-1}}(A^{(2)})$.

According to the first part of Proposition 10, in Δ_i we need to have $key_{\Delta_{i-1}}(A^{(1)}) \cup key_{\Delta_{i-1}}(A^{(2)}) \subseteq key_{\Delta_i}(A)$, which implies that B and C should be on the same root-to-leaf path in Δ_i. Since B and C are already packed in Δ_{i-1}, we infer that they occurred in Δ_{i-1} on the same root-to-leaf path, which contradicts the hypothesis. ◻

Part 2 of Proposition 10. We have to prove that for any variable A it holds that if any two occurrences of A have the same depth in Δ_i for $i \in [m]$, then they are siblings.

Lemma 25 implies that for all Δ_{i-1} ($i \in [m]$), assuming that A is the variable to pack to obtain Δ_i, all occurrences of A in Δ_{i-1} are children of variables from the same root-to-leaf path in Δ_{i-1}. This directly implies the Part 2 of Proposition 10.

D Additional Material for Section 5

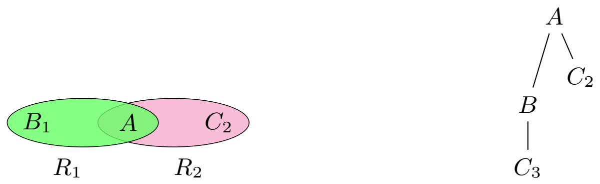
Proof of Proposition 13

We prove that the triangle query has no query plan that is output-bounded for the $\mathcal{T}/\mathcal{F}$-representation systems, and that this also holds for $\mathcal{F}$-representations of the intermediate results and $\mathcal{T}$-representations of the final result.

We recall the triangle query $Q_\triangleleft = R_1(A,B), R_2(A,C), R_3(B,C)$ introduced in Section 2. Since all variables are pairwise dependent, a valid variable order for the result of $Q_\triangleleft$ has a single path and each node has all its ancestors in its key. Consequently, all widths are equal:

$$\rho^*(Q_\triangleleft) = f(Q_\triangleleft) = e(Q_\triangleleft) = \mathit{fhtw}(Q_\triangleleft) = 3/2.$$

Take without loss of generality the plan with join sequence A, B, C.



(a) Hypergraph of the intermediate variable order after joining on A.

(b) Intermediate $\mathcal{F}$-tree after joining on B.

Figure 12 Examples from the proof of Proposition 13 showing that the triangle query is not output-bounded for $\mathcal{T}/\mathcal{F}$-representations.

• If we use $\mathcal{T}$-representations as intermediate results, after the join on A we have an intermediate variable order with a fractional edge cover of 2, obtained after solving the following linear program with positive variables (we depict in Figure 12(a) the corresponding hypergraph):

$$A:\ x_1 + x_2 \geq 1, \qquad B_1:\ x_1 \geq 1, \qquad C_2:\ x_2 \geq 1.$$

Since $2 > 3/2 = \rho^*(Q_\triangleleft)$, the query plan is not output-bounded.

• If we use $\mathcal{F}$-representations as intermediate results, after the join on B we have an intermediate variable order as in Figure 12(b), where each variable has all its ancestors in its key. The intermediate $\mathcal{F}$-tree has a factorization width of 2, obtained after solving the following linear program with positive variables (corresponding to the left branch of the $\mathcal{F}$-tree):

$$A:\ x_1 + x_2 \geq 1, \qquad B:\ x_1 + x_3 \geq 1, \qquad C_3:\ x_3 \geq 1.$$

Since $2 > 3/2 = f(Q_\triangleleft) = \rho^*(Q_\triangleleft)$, the query plan is not output-bounded (for both $\mathcal{T}/\mathcal{F}$-representations of the query result)

For both cases, all other permutations of the join sequence lead to the same conclusion.

Proof of Theorem 15

We prove that every join query has a query plan that is monotonically width-increasing for the $\mathcal{E}/\mathcal{D}$-representation systems.

Take a query plan $(\Delta_0, \ldots, \Delta_m)$ cf. Definition 8 such that either all Δ_i's are $\mathcal{D}$-trees or all Δ_i's are $\mathcal{E}$-trees. To prove that for all $i \in [m]$, it holds that $f(\Delta_{i-1}) \leq f(\Delta_i)$, we need to first show an important auxiliary result.

▶ **Lemma 26.** For every $i \in [m]$, for every not yet packed variable A in Δ_i, for every occurrence $j \in [s]$ of A, there exists a relation whose schema contains $key_{\Delta_i}(A^{(j)}) \cup \{A^{(j)}\}$, if either all Δ_i's are $\mathcal{D}$-trees or all Δ_i's are $\mathcal{E}$-trees. ◻

Proof. The initial Δ_0 is a $\mathcal{T}$-path, and for each of the s occurrences $A^{(j)}$ (for $j \in [s]$) of a variable A in Δ_0, the set $key_{\Delta_0}(A^{(j)})$ contains the set of ancestors that occur in the same relation.

Since we assume $\mathcal{D}$-trees and $\mathcal{E}$-trees, during our query plans the aforementioned characterization of the keys changes only for the variables that are packed. More precisely, if Δ_i is the packing of Δ_{i-1} on A, then $key_{\Delta_i}(A)$ becomes $key_{\Delta_m}(A)$ (cf. Proposition 10).

If A is not yet packed in Δ_{i-1}, for every occurrence $j \in [s]$ it holds that $key_{\Delta_{i-1}}(A^{(j)}) = key_{\Delta_i}(A^{(j)})$ (which moreover is precisely the set of ancestors of $A^{(j)}$ that occur in a same relation). ◻

Now we prove that for all $i \in [m]$, it holds that $f(\Delta_{i-1}) \leq f(\Delta_i)$. We show that for every variable (occurrence) A in Δ_i it holds that $\rho^*(Q_{key_{\Delta_{i-1}}(A)\cup\{A\}}) \leq \rho^*(Q_{key_{\Delta_i}(A)\cup\{A\}})$ by considering three cases:

(i) A was already packed in Δ_{i-1}, for which by Proposition 10 we know that $key_{\Delta_{i-1}}(A) = key_{\Delta_i}(A) = key_{\Delta_m}(A)$. Consequently,

$$\rho^*(Q_{key_{\Delta_{i-1}}(A)\cup\{A\}}) = \rho^*(Q_{key_{\Delta_i}(A)\cup\{A\}}).$$

(ii) A is the variable such that Δ_i is the packing of Δ_{i-1} on A. For each of its occurrences $A^{(j)}$ in Δ_{i-1} (for $j \in [s]$), by Lemma 26 it holds that there exists a relation R such that all variables in $key_{\Delta_{i-1}}(A^{(j)})$ appear in the schema of R, and consequently

$$\rho^*(Q_{key_{\Delta_{i-1}}(A^{(j)})\cup\{A^{(j)}\}}) = 1 \leq \rho^*(Q_{key_{\Delta_i}(A)\cup\{A\}}),$$

since the value of the parameter ρ^* cannot be smaller than 1 by definition.

(iii) A remains not yet packed in Δ_i. By Lemma 26, for each of its s occurrences, there exists a relation R such that all variables in $key_{\Delta_{i-1}}(A^{(j)})$ appear in the schema of R, and moreover all variables in $key_{\Delta_i}(A^{(j)})$ appear in the schema of R. Consequently, $\rho^*(Q_{key_{\Delta_{i-1}}(A^{(j)})\cup\{A^{(j)}\}}) = \rho^*(Q_{key_{\Delta_i}(A^{(j)})\cup\{A^{(j)}\}}) = 1$ for each occurrence $j \in [s]$.

In conclusion, for every variable (occurrence) A in Δ_i we have $\rho^*(Q_{key_{\Delta_{i-1}}(A)\cup\{A\}}) \leq \rho^*(Q_{key_{\Delta_i}(A)\cup\{A\}})$, which implies that $f(\Delta_{i-1}) \leq f(\Delta_i)$.

From the first part of the theorem, and the relationships between the four representation systems and their width measures (cf. Figure 1), we infer that there are also monotonically width-increasing plans for : $\mathcal{E}$-representations of the intermediate results and $\mathcal{T}/\mathcal{F}$-representations of the final result; and for $\mathcal{D}$-representations of the intermediate results and $\mathcal{T}/\mathcal{F}/\mathcal{E}$-representations of the final result.

Proof of Proposition 16

We prove that the Path7 query (Example B.3) has no query plan that is output-bounded for $\mathcal{E}$-representations of the intermediate results and $\mathcal{D}$-representations of the final result.

Since the query is acyclic, we have $\mathit{fhtw}(\mathsf{Path7}) = 1$. Any plan for Path7 consisting only of $\mathcal{E}$-trees cannot avoid an intermediate variable order with e-width of 2 (since an optimal $\mathcal{E}$-tree for Path7 has an e-width of 2 cf. Example B.3).

E Additional Material for Section 6

E.1 Omitted Proofs

Proof of Theorem 19

The theorem to prove is: Given a step in a query plan, where the input is a $\mathcal{D}$-representation IN over $\mathcal{D}$-tree Δ and A is the variable packed at Δ, Joen computes a $\mathcal{D}$-representation OUT of the join result over a $\mathcal{D}$-tree that is a packing of Δ on A in time $O(|\text{IN}| + |\text{OUT}|)$. ☐

First, we analyze the complexity of Step 4, because proving this step is enough to infer that the result holds when we restrict ourselves to $\mathcal{E}$-representations. Afterwards, we analyze Step 1, 2, 3, and 5 to conclude the linear time behavior for general $\mathcal{D}$-representations.

The Joen algorithm (cf. Figure 7) does precisely one pass on the input $\mathcal{E}$-representation IN. Next, we analyze the number of computation steps that Joen needs to create the output $\mathcal{E}$-representation OUT.

To this purpose, we first characterize the number of data values of a given variable after packing it with Joen (Lemma 27) and the number of computation steps needed for a Joen application (Lemma 28).

Before analyzing the Joen complexity, we recall a notation introduced in Section 6.2: by $\eta_{K_p}[k_1, \ldots, k_{p-1}]$ we denote the list of K_p-values under the K_{p-1}-value k_{p-1}, ..., under the K_1-value k_1.

▶ **Lemma 27.** Given a factorization over Δ_{i-1} and the variable A such that the packing of Δ_{i-1} on A yields the variable order Δ_i, with $key_{\Delta_i}(A) = \{K_1, \ldots, K_p\}$, the number of A-values in the factorization over Δ_i is:

$$\sum_{k_1 \in \eta_{K_1}} \cdots \sum_{k_p \in \eta_{K_p}[k_1, \ldots, k_{p-1}]} = |\eta_A[k_1, \ldots, k_p]|.$$

Proof. The number follows from Lemma 7.5 in [13], which characterizes the number of data values in a factorization. ◻

Next, we analyze the number of computation steps for a Joen application.

▶ **Lemma 28.** Given a factorization over Δ_{i-1} and the variable A such that the packing of Δ_{i-1} on A yields the variable order Δ_i, with $key_{\Delta_i}(A) = \{K_1, \ldots, K_p\}$, the number of Joen computation steps is:

$$\sum_{k_1 \in \eta_{K_1}} \cdots \sum_{k_p \in \eta_{K_p}[k_1, \ldots, k_{p-1}]} \min\{|\eta_{A_1}[k_1^1, \ldots, k_{l_1}^1]|, \ldots, |\eta_{A_s}[k_1^l, \ldots, k_{l_s}^l]|\},$$

where for $j \in [s]$, by $\{k_1^j, \ldots, k_{l_j}^j\}$ we denote the subset of $\{k_1, \ldots, k_p\}$ restricted to the values corresponding to variables from $key_{\Delta_{i-1}}(A^{(j)})$. ◻

Proof. For every combination of values $k_1, \ldots, k_p$, we need to intersect the s lists of $A^{(j)}$-values ($j \in [s]$) from the initial factorization over Δ_{i-1}.

We infer that every list of $A^{(j)}$-values depends on a subset of $\{k_1, \ldots, k_p\}$ (that we denote $\{k_1^j, \ldots, k_{l_j}^j\}$) because of the key preservation property. Moreover, since all the lists of $A^{(j)}$-values are ordered, the number of computation steps needed to intersect them is equal to the minimal length of all these lists.

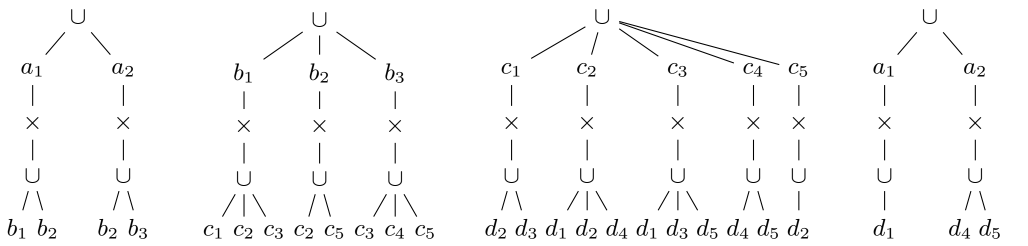
We measure only the intersection time because every A-value from the result factorization over Δ_i inherits the pointers to the children of $A_1, \ldots, A_s$, respectively, hence no computation is performed below the intersected lists. ◻

Lemma 27 and 28 show that creating the Joen result takes time linear in the size of the output, which is sufficient to say that Step 3 takes the desired amount of time (hence the Theorem holds when restricted to $\mathcal{E}$-representations).

Before proving that the time bound holds also for Step 1, 2, and 4 to conclude the proof of Theorem 19, we prove an auxiliary property of linear programs.

▶ **Lemma 29.** Given a linear program over a set X of positive variables and an arbitrary number of inequalities $\sum_{y \in Y} y \geq 1$ over subsets $Y \subseteq X$ and whose goal is to minimize $\sum_{x \in X} x$. The optimal solution of the program is an upper bound for the optimal solution of any program obtained by removing some of the inequalities. ◻

Proof. Let m be the number of inequalities and s be the optimal solution of the linear program. Assume w.l.o.g. that we remove the last $m - k$ inequalities and we keep the first



(a) $R_1(A_1, B_1)$. **(b)** $R_2(B_2, C_2)$. **(c)** $R_3(C_3, D_3)$. **(d)** $R_4(A_4, D_4)$.

Figure 13 Input database for the Loop4 query.

k inequalities. Take the subset of variables that appear in the first k inequalities and give them the same values as in the optimal solution of the initial linear program. Hence, (i) all first k inequalities are satisfied (in other words the solution to the initial problem is still a solution for the new program), and (ii) their sum is bounded by s. The part (ii) holds because all variables are assigned to positive numbers. $\square$

In Step 1, projecting away all values that are not for variables in the key can be done in one pass over the database. Then, in Step 2, merging the definitions that now have the same context is also in linear time in the input (modulo a log factor): we concatenate the definitions that have the same context, we sort the result, and then remove duplicates. The size of the output of Step 3 is asymptotically bounded by the size of the Joen output due to Lemma 29 and the fact that the linear program for the leaf A of the Joen output contains precisely the same inequalities as in the linear program for the leaf K_p in the Joen input and an additional inequality for A. Consequently, the time to produce it is also bounded by the size of the final result. As for Step 5, everything can be done in a single (bottom-up) pass (in particular the cleanup step as in [13] and as recalled in Section 6.1).

E.2 Example of Joen on $\mathcal{D}$-representations

▶ **Example 30.** Take the query Loop4 defined as follows:

$$R_1(A, B), R_2(B, C), R_3(C, D), R_4(A, D).$$

We recall that we depicted the query plan consisting of $\mathcal{D}$-trees in Figure 5. We depict the input database in Figure 13. Take the variable order $A\{B\{C\{D\}\}\}$, where $key(B) = \{A\}$, $key(C) = \{A, B\}$, and $key(D) = \{A, C\}$. We recall that $A \in key(C)$ although A and C do not occur in a same relation because $A \in key(D)$ and D is a child of C in the variable order.

Assume that we have already done the joins on A, B and C, and now we want to join on D. Before joining on D, the keys for each of its occurrences consist of their ancestors in the variable order that appear in a same input relation. More precisely, $key(D_3) = \{C\}$ and $key(D_4) = \{A\}$. The current $\mathcal{D}$-representation is:

$$\eta_A = \{a_1, a_2\}, \quad \eta_B[a_1] = \{b_1, b_2\}, \quad \eta_B[a_2] = \{b_2, b_3\},$$
$$\eta_C[a_1, b_1] = \{c_1, c_2, c_3\}, \quad \eta_C[a_1, b_2] = \{c_2, c_5\}, \quad \eta_C[a_2, b_2] = \{c_2, c_5\}, \quad \eta_C[a_2, b_3] = \{c_3, c_4, c_5\},$$
$$\eta_{D_4}[a_1] = \{d_1\}, \quad \eta_{D_4}[a_2] = \{d_4, d_5\}, \quad \eta_{D_3}[c_1] = \{d_2, d_3\}, \quad \eta_{D_3}[c_2] = \{d_1, d_2, d_4\},$$
$$\eta_{D_3}[c_3] = \{d_1, d_3, d_5\}, \quad \eta_{D_3}[c_4] = \{d_4, d_5\}, \quad \eta_{D_3}[c_5] = \{d_2\}.$$

There are two definitions with the same mapping $\{c_2, c_5\}$, which is the result of the intersection of the list of C's under b_2 in R_2 with the list of C's in R_3. This list is stored twice i.e., for every A-value paired with b_2 in the intermediate result because $A \in \mathit{key}(C)$ as dictated by the final variable order.

Step 1 and 2. We project away the values for the variable B (which is not in $\mathit{key}(D)$) and aggregate the definitions that have the same context and variable:

$$F_A = \{a_1, a_2\}, \quad F_C[a_1] = \{c_1, c_2, c_3, c_5\}, \quad F_C[a_2] = \{c_2, c_3, c_4, c_5\}.$$

Step 3. We construct a factorization over the $\mathcal{E}$-tree consisting of the path $A\{C\}$ and keeping pointers to the original lists of D_3's and D_4's, respectively.

Step 4. We run the Joen algorithm (cf. Figure 7), which returns a factorization over the $\mathcal{E}$-tree $A\{C\{D\}\}$ (where each variable has all its ancestors in its key). In particular for D we construct:

$$\eta_D[a_1, c_1] = \emptyset, \quad \eta_D[a_1, c_2] = \{d_1\}, \quad \eta_D[a_1, c_3] = \{d_1\}, \quad \eta_D[a_1, c_5] = \emptyset$$
$$\eta_D[a_2, c_2] = \{d_4\}, \quad \eta_D[a_2, c_3] = \{d_5\}, \quad \eta_D[a_2, c_4] = \{d_4, d_5\}, \quad \eta_D[a_2, c_5] = \emptyset$$

Step 5. We remove all definitions η_{D_3} and η_{D_4}, and we add instead the non-empty η_D from the previous step. We also remove the values c_1 and c_5 since they only appear in definitions with empty unions. We thus obtain:

$$\eta_A = \{a_1, a_2\}, \quad \eta_B[a_1] = \{b_1, b_2\}, \quad \eta_B[a_2] = \{b_2, b_3\},$$
$$\eta_C[a_1, b_1] = \{c_2, c_3\}, \quad \eta_C[a_1, b_2] = \{c_2\}, \quad \eta_C[a_2, b_2] = \{c_2\}, \quad \eta_C[a_2, b_3] = \{c_3, c_4\},$$
$$\eta_D[a_1, c_2] = \{d_1\}, \quad \eta_D[a_1, c_3] = \{d_1\},$$
$$\eta_D[a_2, c_2] = \{d_4\}, \quad \eta_D[a_2, c_3] = \{d_5\}, \quad \eta_D[a_2, c_4] = \{d_4, d_5\} \qquad \square$$