Covers of Query Results

Ahmet Kara
Department of Computer Science, University of Oxford, Oxford, UK
ahmet.kara@cs.ox.ac.uk

Dan Olteanu
Department of Computer Science, University of Oxford, Oxford, UK
dan.olteanu@cs.ox.ac.uk

Abstract

We introduce succinct lossless representations of query results called covers. They are subsets of the query results that correspond to minimal edge covers in the hypergraphs of these results.

We first study covers whose structures are given by fractional hypertree decompositions of join queries. For any decomposition of a query, we give asymptotically tight size bounds for the covers of the query result over that decomposition and show that such covers can be computed in worst-case optimal time up to a logarithmic factor in the database size. For acyclic join queries, we can compute covers compositionally using query plans with a new operator called cover-join. The tuples in the query result can be enumerated from any of its covers with linearithmic pre-computation time and constant delay.

We then generalize covers from joins to functional aggregate queries that express a host of computational problems such as aggregate-join queries, in-database optimization, matrix chain multiplication, and inference in probabilistic graphical models.

2012 ACM Subject Classification Information systems → Relational database model, Information systems → Join algorithms, Information systems → Relational database query languages

Keywords and phrases factorized database, fractional hypertree decomposition, functional aggregate query, minimal edge cover, query plan

Digital Object Identifier 10.4230/LIPIcs.ICDT.2018.16

Funding This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement 682588.

Acknowledgements The authors would like to thank Milos Nikolic, Max Schleich, and the anonymous reviewers for their feedback on drafts of this paper, and Yu Tang for inspiring discussions that led to the concept of cover as a relational alternative to factorized representations.

1 Introduction

This paper introduces succinct lossless representations of query results called covers. Given a database and a join query or, more generally, a functional aggregate query (FAQ) [18], a cover is a subset of the query result that, together with a (fractional hypertree) decomposition of the query [13], recovers the query result. Covers enjoy desirable properties.

First, they can be more succinct than the listing representation of the query result. For a join query Q, database $\mathbf{D}$, and a decomposition $\mathcal{T}$ of Q with fractional hypertree width w [20], a cover over $\mathcal{T}$ has size $\mathcal{O}(|\mathbf{D}|^w)$. In contrast, there are arbitrarily large databases for which the listing representation of the query result has size $\Omega(|\mathbf{D}|^{\rho^*})$, where ρ^* is the fractional edge cover number of Q [4]. The gap between the fractional hypertree width and the fractional edge cover number can be as large as the number of relation symbols in Q.

For an FAQ (and the special case of a join query) φ, any cover of its result can be computed in time $\mathcal{O}(|\mathbf{D}|^w \log |\mathbf{D}|)$, where w is the FAQ-width [18] of φ. FAQs can express aggregates over database joins [6], in-database optimization [24, 2], matrix chain multiplication, and inference in probabilistic graphical models.

Second, the tuples in the query result can be enumerated from one of its covers with linearithmic pre-computation time and constant delay. This is not the case for the representation defined by the pair of database and join query (unless W[1]=FPT) [25]. The benefits of covers over the latter representation are less apparent for acyclic queries, for which both representations share the same linear-size bound and desirable enumeration complexity [5]. For acyclic joins, the question thus becomes why to succinctly represent a query result by one relation instead of the pair of a set of relations and the query. We next highlight three practical benefits. Covers readily provide a subset of the query result without the need to compute the join. This improves cache locality for subsequent operations, e.g., aggregates, since we only need to read in tuple by tuple from the cover instead of reading tuples from different relations stored at different locations in memory and then joining them. Similarly, covers provide access locality for disk operations since tuples from the cover are stored on the same disk page, whereas tuples from different relations are stored on different pages. Furthermore, covers are samples of the query result that disregard the uninformative yet exhaustive pairings brought by Cartesian products. In exploratory data analysis, the explicit listing of Cartesian products is overwhelming to the user since it may be very large. An alternative approach that would present the user with many relations and the query, would have to rely on the user to figure out possible tuples in the query result, which is not desirable. A cover, in contrast, is a compact relation that absolves the user from ad-hoc joining of relations and from re-discovering Cartesian products in a large listing of tuples. Finally, processing following the in-database joins may require a single relation as input, as it is the case for machine learning over joins [24]. Indeed, instead of learning regression models over the result of a join we can instead learn them over one of its covers.

Third, covers use the standard listing representation. Prior work introduced lossless representations of query results called *factorized databases* that achieve the same succinctness as covers, yet they are directed acyclic graphs that represent the query result as circuits whose nodes are data values or the relational operators Cartesian product and union [23]. The graph representation makes difficult their adoption as a data representation model by mainstream database systems that rely on relational storage (factorized *computation* is however used in relational systems [2]). A relational alternative to factorized databases, as metamorphosed in covers, can prove useful in a variety of settings. The intermediate results in query plans can be represented as covers. In distributed query plans, covers can encode succinctly the otherwise expensive intermediate query results that are communicated among servers in each round [26] and can be processed as soon as each of their tuples is received.

The contributions of this paper are as follows:

- Section 3 introduces covers of join query results and their correspondence to minimal edge covers in the hypergraphs of the query results. We also give tight size bounds for covers and show that the tuples in the query result can be enumerated from any cover with linearithmic pre-computation time and constant delay.
- Given a database and a join query, covers of its result can be computed in worst-case optimal time (modulo a log factor). Section 4 focuses on the compositionality of cover computation for acyclic join queries. We introduce cover-join plans to compute covers in time linearithmic in their sizes and the size of the input database. A cover-join plan is a binary plan that follows the structure of a join tree of the acyclic query. It uses a

cover-join operator that computes covers of the join of two relations, which may be input relations or covers for subqueries. Different plans may lead to different sets of covers. There are covers that cannot be obtained using binary plans.

- Section 5 generalizes our notion of covers from joins to functional aggregate queries by representing succinctly both tuples and aggregates in the query result.

We consider natural join queries where each relation is used at most once. The appendix extends our results to arbitrary equi-join queries and provides further details and examples. The proofs of the formal statements are given in an extended version of this paper [17].

Related work. There are three strands of directly related work: cores in databases and graph theory; succinct representations of query results; and normal forms for relational data.

Cores of graphs, queries, and universal solutions to data exchange problems revolve around smaller yet lossless representations that are homomorphically minimal subgraphs [16], subqueries [8], and universal solutions [11], respectively. A further application of graph cores is in the context of the Semantic Web, where cores of RDF graphs are used to obtain minimal representations and normal forms of such graphs [15]. Our notion of covers is different. Covers rely on query decompositions to achieve succinctness, and they only become lossless *in conjunction* with a decomposition. If we ignore the decomposition, the covers become lossy as they are subsets of the result. Whereas in data exchange all universal solutions have the same core (up to isomorphism), the result of a query may have exponentially many incomparable covers. While not a defining component of cores in data exchange, generalized hypertree decompositions can help derive improved algorithms for computing the core of a relational instance with labeled nulls under different classes of dependencies [12].

Covers are relational encodings of d-representations, a lossless graph-based factorization of the query result [23]. The structure of d-representations is given by variable orders called d-trees, which are an alternative syntax for fractional hypertree decompositions. Whereas d-representations are lossless on their own, covers need the decomposition to derive the missing tuples. Decompositions are the data-independent price to pay for achieving the data-dependent succinctness of factorized representations using the listing representation. Both d-representations and covers achieve succinctness by avoiding the materialization of Cartesian products. Whereas the former encode the products symbolically and losslessly, the covers only keep a minimal subset of the product that is enough to reconstruct it entirely.

The goal of database design is to avoid redundancy in the *input* database. Existing normal forms achieve this by *decomposing* one relation into several relations guided by functional and join dependencies [9]. Covers exploit the join dependencies to avoid redundancy in the query *output*. They do not decompose the result back into the (now globally consistent) input database. Like factorized representations, covers are a normal form for relations representing query results. From a cover of a join result over a decomposition, we can obtain a decomposition of the join result in project-join normal form (5NF) [10] by taking one projection of the cover onto the attributes of each bag of the decomposition.

2 Preliminaries

Databases. We assume an ordered domain of data values. A relation schema is a finite set of attributes. For an attribute A, we denote by $\mathsf{dom}(A)$ its domain. A database schema is a finite set of relation symbols. A tuple t over a relation schema S is a mapping from the attributes in S to values in their respective domains. A relation over a relation schema S is a finite set of tuples over S. A database $\mathbf{D}$ over a database schema $\mathcal{S}$ contains for each relation symbol in $\mathcal{S}$, a relation over the same schema. For a relation (symbol) R and tuple t, we use $\mathcal{S}(R)$ and $\mathcal{S}(t)$ to refer to their schemas and write $R(S)$ to express that the schema of R is S. The tuples $t_1, \ldots, t_n$ are joinable if $\pi_{S_{i,j}} t_i = \pi_{S_{i,j}} t_j$ for all $i, j \in [n]$ and $S_{i,j} = \mathcal{S}(t_i) \cap \mathcal{S}(t_j)$. The size $|R|$ of a relation R is the number of its tuples. The size $|\mathbf{D}|$ of a database $\mathbf{D}$ is the sum of the sizes of its relations.

Natural Join Queries. We consider natural join queries of the form $Q = R_1(S_1) \bowtie \ldots \bowtie R_n(S_n)$, where each R_i is a relation symbol over relation schema S_i and refers to a database relation over the same schema. Notation-wise we do not distinguish between a relation symbol and the corresponding relation. The joins in Q are expressed by sharing attributes across relation schemas. The schema $\mathcal{S}(Q)$ of Q is the set of relation symbols in Q: $\mathcal{S}(Q) = \{R_i\}_{i \in [n]}$. The set $att(Q)$ of attributes of Q is the union of the schemas of its relation symbols: $att(Q) = \bigcup_{i \in [n]} S_i$. The size $|Q|$ of Q is the number of its relation symbols: $|Q| = n$. A database is *globally consistent* with respect to a query Q if there are no (dangling) tuples that do not contribute to the result of Q [1]. Two relations R_1 and R_2 are *consistent* if the database $\{R_1, R_2\}$ is globally consistent with respect to the query $R_1 \bowtie R_2$. We assume that the relation symbols in Q are non-repeating and each relation symbol corresponds to a distinct relation. Appendix C extends our contributions to arbitrary equi-join queries.

Hypergraphs. Let H be a multi-hypergraph (hypergraph for short) whose edge multiset E may contain multiple hyperedges (edges for short) with the same node set. A fractional edge cover for H is a function γ mapping each edge in H to a positive number such that $\Sigma_{e \ni v} \gamma(e) \geq 1$ for each node v of H, i.e., the sum of the function values for all edges incident to v is at least 1. We define the weight of a fractional edge cover γ as $weight(\gamma) = \Sigma_{e \in E} \gamma(e)$. The fractional edge cover number $\rho^*(H)$ of H is the minimum weight of fractional edge covers of H. It can be obtained from a fractional edge cover where the edge weights are rational numbers of bit-length polynomial in the size of H [4].

We use hypergraphs for queries and for relations representing their results. The hypergraph H of a query Q consists of one node A for each attribute A in Q and one edge $\mathcal{S}(R)$ for each relation symbol $R \in \mathcal{S}(Q)$. We define $\rho^*(Q) = \rho^*(H)$.

Let R be a relation and $\mathcal{P}$ a set of (possibly overlapping) subsets of $\mathcal{S}(R)$ such that $\bigcup_{S \in \mathcal{P}} S = \mathcal{S}(R)$. The hypergraph H of R over $\mathcal{P}$ consists of one node for each distinct tuple in $\pi_S R$ for each attribute set $S \in \mathcal{P}$ and one edge for each tuple in R. The edge for a tuple t thus consists of all nodes for tuples $\pi_S(t)$ with $S \in \mathcal{P}$. We use $tuple(v)$ to denote the tuple represented by a node or edge v in H. Given a subset M of the edges in H, we define $rel(M) = \{tuple(e)\}_{e \in M}$ as the relation represented by M. The set M is an edge cover of H if each node in H is contained in at least one edge in M. The set M is a minimal edge cover if it is an edge cover and any of its strict subsets is not.

▶ **Example 1.** Consider the path query $Q = R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(C, D)$. Figure 1 depicts in the top row a database of the three relations R_1, R_2 and R_3, the query result and a subset of it. In the bottom row, the figure depicts the hypergraph of Q (and its decomposition defined below), the hypergraph of its result over the attribute sets $\{\{A, B\}, \{B, C\}, \{C, D\}\}$, and the hypergraph of a subset of the query result over the same attribute sets.

Decompositions. A *hypertree decomposition* $\mathcal{T}$ of (the hypergraph H of) a query Q is a pair (T, χ), where T is a tree and χ a function mapping each node in T to a subset of the nodes of H. For a node $t \in T$, the set $\chi(t)$ is called a bag. A hypertree decomposition satisfies

R_1

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_2	b_2
a_1	b_3

R_2

B	C
b_1	c_1
b_2	c_2
b_3	c_3
b_4	c_4

R_3

C	D
c_1	d_1
c_1	d_2
c_2	d_1
c_2	d_2
c_4	d_1

$Q(\mathbf{D})$

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_1	d_2
a_2	b_1	c_1	d_1
a_2	b_1	c_1	d_2
a_1	b_2	c_2	d_2
a_1	b_2	c_2	d_1
a_2	b_2	c_2	d_2
a_2	b_2	c_2	d_1

$rel(M)$

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_1	c_1	d_2
a_1	b_2	c_2	d_1
a_2	b_2	c_2	d_2

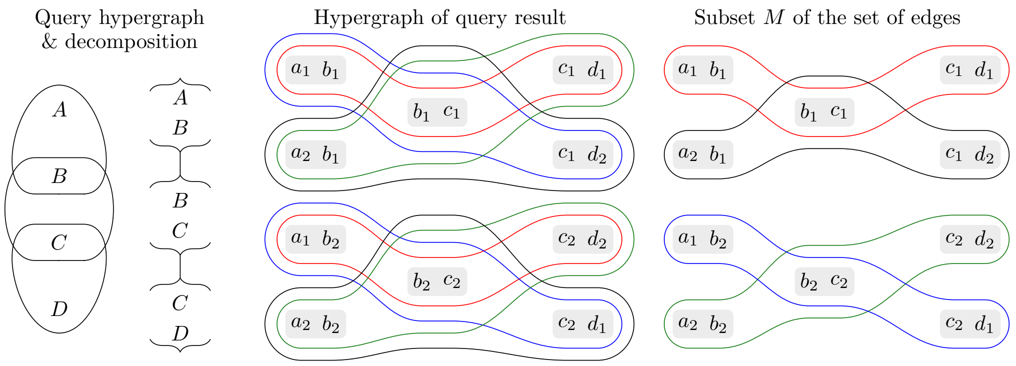

Query hypergraph & decomposition
Hypergraph of query result
Subset M of the set of edges
A
B
C
D

Figure 1 Top row: database $\mathbf{D} = \{R_1, R_2, R_3\}$, the result $Q(\mathbf{D})$ of the path query Q in Example 1, and a subset of $Q(\mathbf{D})$; bottom row: the hypergraph of Q, the tree of a decomposition $\mathcal{T}$ of Q, the hypergraph of $Q(\mathbf{D})$ over attribute sets $\mathcal{S}(\mathcal{T})$, and a minimal edge cover M of this hypergraph.

two properties. *Coverage*: For each edge e in H, there must be a node t in T with $e \subseteq \chi(t)$. *Connectivity*: For each node v in H, the set $\{t \mid t \in T, v \in \chi(t)\}$ must be non-empty and form a connected subtree in T. The schema of $\mathcal{T}$ is the set of its bags: $\mathcal{S}(\mathcal{T}) = \{\chi(t) \mid t \in T\}$. The attributes of $\mathcal{T}$ are defined by $att(\mathcal{T}) = \bigcup_{B \in \mathcal{S}(\mathcal{T})} B$.

A *fractional hypertree decomposition* [14] of (the hypergraph H of) a query Q is a triple $(T, \chi, \{\gamma_t\}_{t \in T})$ where (T, χ) is a hypertree decomposition of H and for each node $t \in T$, γ_t is a fractional edge cover of minimal weight for the subgraph of H restricted to $\chi(t)$. We define the *fractional hypertree width* of $\mathcal{T} = (T, \chi, \{\gamma_t\}_{t \in T})$ as $\max_{t \in T}\{weight(\gamma_t)\}$ and we denote it by $\mathsf{fhtw}(\mathcal{T})$. The fractional hypertree width $\mathsf{fhtw}(H)$ of the hypergraph H is the minimal possible such width of any fractional hypertree decomposition of H. The fractional hypertree width $\mathsf{fhtw}(Q)$ of a query Q is the fractional hypertree width $\mathsf{fhtw}(H)$ of its hypergraph H. For simplicity, we use the terms decomposition and width in place of fractional hypertree decomposition and fractional hypertree width, respectively.

A hypergraph H is α*-acyclic* (acyclic for short) if it has a decomposition in which each bag is contained in an edge of H [7]. A query whose hypergraph is acyclic is also called acyclic. The width of any acyclic hypergraph or query is one. A *join tree* of a query Q is a labelled tree (T, ℓ) where $T = (\mathcal{S}(Q), E)$ is a tree and ℓ is an edge labelling such that (i) each edge $e = (R, R') \in E$ is labelled by $\ell(e) = \mathcal{S}(R) \cap \mathcal{S}(R')$ and (ii) for every pair R, R' of distinct nodes and for each attribute $A \in \mathcal{S}(R) \cap \mathcal{S}(R')$, the label of each edge along the unique path between R and R' includes A (Section 6.4 in [1]). A query is acyclic if and only if it admits a join tree (Theorem 6.4.5 in [1]). The decomposition $\mathcal{T}$ *corresponding* to the join tree $\mathcal{J}$ of a query Q is constructed as follows. Each node in $\mathcal{J}$, which corresponds to a relation symbol R, is mapped to a node in $\mathcal{T}$, which has the bag $\mathcal{S}(R)$. For each node t in $\mathcal{T}$ with bag $\mathcal{S}(R)$, the function γ_t maps the hyperedge for R to 1.

▶ **Example 2.** Figure 1 gives the hypergraph (left, bottom row) of the path query in Example 1 along with one of its decompositions. This decomposition has width one, since each bag is included in one edge of the hypergraph; the path query is acyclic. The decomposition, where the top two bags are merged into one, has width two. For queries with cycles, the width can be larger than one. For instance, the width of the triangle query is 3/2 [4].

Computational Model. We use the uniform-cost RAM model [3] where data values as well as pointers to databases are of constant size. Our analysis is with respect to data complexity where the query is assumed fixed. We use $\widetilde{\mathcal{O}}$ to hide a $\log |\mathbf{D}|$ factor.

Result-preserving Transformation. Let $(Q, \mathcal{T}, \mathbf{D})$ denote a triple of a natural join query Q, a decomposition $\mathcal{T}$ of Q, and a database $\mathbf{D}$.

▶ **Proposition 3.** *Given $(Q, \mathcal{T}, \mathbf{D})$, we can compute $(Q', \mathcal{T}, \mathbf{D}')$ with size $\mathcal{O}(|\mathbf{D}|^{\mathsf{fhtw}(\mathcal{T})})$ and in time $\widetilde{\mathcal{O}}(|\mathbf{D}|^{\mathsf{fhtw}(\mathcal{T})})$ such that Q' is an acyclic natural join query, $\mathcal{T}$ corresponds to a join tree of Q', $\mathbf{D}'$ is globally consistent with respect to Q' and $Q'(\mathbf{D}') = Q(\mathbf{D})$.*

▶ **Example 4.** Consider the path query Q, decomposition $\mathcal{T}$, and database $\mathbf{D}$ in Example 2. The application of Proposition 3 leaves Q unchanged, since Q is already acyclic and $\mathcal{T}$ corresponds to a join tree of Q. The database in Figure 1 is not globally consistent with respect to Q, since it contains tuples (under the thin lines) that do not contribute to the result. We remove these dangling tuples to make it consistent.

Consider now the bowtie query $Q_{\bowtie} = R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(A, C) \bowtie R_4(A, D) \bowtie R_5(D, E) \bowtie R_6(A, E)$. A decomposition $\mathcal{T}_{\bowtie}$ with the lowest width of 3/2 has two bags $S_1 = \{A, B, C\}$ and $S_2 = \{A, D, E\}$, one for each clique (triangle) in the query. The application of Proposition 3 constructs the acyclic query $Q' = B_1(A, B, C) \bowtie B_2(A, D, E)$. The relations $B_1(A, B, C)$ and $B_2(A, D, E)$ are materializations of the two bags of $\mathcal{T}_{\bowtie}$. The database $\mathbf{D}' = \{B_1(A, B, C), B_2(A, D, E)\}$ is globally consistent with respect to Q', i.e., each tuple in B_1' has at least one joinable tuple in B_2' and vice versa. The decomposition $\mathcal{T}_{\bowtie}$ corresponds to a join tree of Q'.

3 Covers for Join Queries

In this section we introduce the notion of covers of join query results along with a characterization of their size bounds, the connection to minimal edge covers for hypergraphs of join query results, and the complexity for enumerating the tuples in the query result from a cover.

Let $(Q, \mathcal{T}, \mathbf{D})$ denote a triple of a natural join query Q, decomposition $\mathcal{T}$ of Q, and database $\mathbf{D}$. For an instance $(Q, \mathcal{T}, \mathbf{D})$, covers of the query result $Q(\mathbf{D})$ are relations that are minimal while preserving the information in the query result $Q(\mathbf{D})$ in the following sense.

▶ **Definition 5** (Result Preservation)**.** A relation K is *result-preserving with respect to* $(Q, \mathcal{T}, \mathbf{D})$ if its schema $\mathcal{S}(K)$ is $att(Q)$ and $\pi_B K = \pi_B Q(\mathbf{D})$ for each $B \in \mathcal{S}(\mathcal{T})$.

That is, for each bag B in the decomposition $\mathcal{T}$ of Q, both the relation K and the query result $Q(\mathbf{D})$ have the same projection onto B. This also means that the natural join of these projections of K is precisely $Q(\mathbf{D})$.

▶ **Proposition 6.** *Given $(Q, \mathcal{T}, \mathbf{D})$, a relation K with schema $att(Q)$ is result-preserving with respect to $(Q, \mathcal{T}, \mathbf{D})$ if and only if $\bowtie_{B \in \mathcal{S}(\mathcal{T})} \pi_B K = Q(\mathbf{D})$.*

We further say that the relation K is *minimal* result-preserving with respect to $(Q, \mathcal{T}, \mathbf{D})$ if it is result-preserving with respect to $(Q, \mathcal{T}, \mathbf{D})$, yet this is not the case for any strict subset of it. We can now define the notion of covers of query results.

▶ **Definition 7** (Covers)**.** Given $(Q, \mathcal{T}, \mathbf{D})$, a *cover of the query result* $Q(\mathbf{D})$ *over the decomposition* $\mathcal{T}$ is a minimal result-preserving relation with respect to $(Q, \mathcal{T}, \mathbf{D})$.

▶ **Example 8.** Figure 1 gives the decomposition $\mathcal{T}$ of a path query and one cover *rel*(M) of the query result over $\mathcal{T}$. We give below four relations that are subsets of the query result. The relations K_1 and K_2 are covers, while the relations N_1 and N_2 are not covers:

K_1

A	B	C	D
a_1	b_1	c_1	d_2
a_2	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_2	b_2	c_2	d_1

K_2

A	B	C	D
a_1	b_1	c_1	d_2
a_2	b_1	c_1	d_1
a_1	b_2	c_2	d_1
a_2	b_2	c_2	d_2

N_1

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_1	d_2
a_1	b_2	c_2	d_1

N_2

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_1	d_2
a_2	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_2	c_2	d_1

To check the minimal result-preservation property, we take projections onto the bags $B_1 = \{A, B\}$, $B_2 = \{B, C\}$, and $B_3 = \{C, D\}$. The relation N_1 is not result-preserving, because $(a_2, b_2) \notin \pi_{B_1} N_1$. The same argument also applies to relation N_2.

Consider now the coarser decomposition $\mathcal{T}'$ with bags $B'_{1,2} = \{A, B, C\}$ and $B'_3 = \{C, D\}$. The covers over $\mathcal{T}$ discussed above are also covers over $\mathcal{T}'$. The query result is the only cover over the coarsest decomposition $\mathcal{T}''$ with only one bag.

▶ **Example 9.** A query result may admit exponentially many covers over the same decomposition. Consider for instance the product query $R_1(A) \bowtie R_2(B)$ with relations R_1 and R_2 of size two and respectively $n > 1$. The query result has size $2 \cdot n$. To compute a cover, we pair the first tuple in R_1 with any non-empty and strict subset of the n tuples in R_2, while the second tuple in R_1 is paired with the remaining tuples in R_2. There are $2^n - 2$ possible covers. The empty and the full sets are missing from the choice of a subset of R_2 as they would mean that one of the two tuples in R_1 would have to be paired with tuples in R_2 that are already paired with the other tuple in R_1 and that would violate the minimality criterion of the covers. All covers have size n and none is contained in another.

We next give a characterization of covers via the hypergraph of the query result.

▶ **Proposition 10.** *Given* $(Q, \mathcal{T}, \mathbf{D})$*, a relation* K *is a cover of the query result* $Q(\mathbf{D})$ *over* $\mathcal{T}$ *if and only if the hypergraph of* $Q(\mathbf{D})$ *over* $\mathcal{S}(\mathcal{T})$ *has a minimal edge cover* M *such that* *rel*$(M) = K$*.*

▶ **Example 11.** Figure 1 gives a minimal edge cover M and the cover *rel*(M). By removing any edge from M, it is not anymore an edge cover. By removing the tuple corresponding to that edge from *rel*(M), it is not anymore a cover since it is not result preserving. By adding an edge to M or the corresponding tuple to *rel*(M), they are not anymore minimal.

We now turn our investigation to sizes and first note the following immediate property.

▶ **Proposition 12.** *Given* $(Q, \mathcal{T}, \mathbf{D})$*, each cover of* $Q(\mathbf{D})$ *over* $\mathcal{T}$ *is a subset of* $Q(\mathbf{D})$*.*

An implication of Proposition 12 is that the covers cannot be larger than the query result. However, they can be much more succinct. We first give size bounds for covers using the sizes of projections of the query result onto the bags of the underlying decomposition.

▶ **Proposition 13.** *Given $(Q, \mathcal{T}, \mathbf{D})$, the size of each cover K of $Q(\mathbf{D})$ over $\mathcal{T}$ satisfies the inequalities* $\max_{B\in\mathcal{S}(\mathcal{T})}\{|\pi_B Q(\mathbf{D})|\} \leq |K| \leq \Sigma_{B\in\mathcal{S}(\mathcal{T})} |\pi_B Q(\mathbf{D})|$.

We can now characterize the size of a cover using the width of the decomposition.

▶ **Theorem 14.** *Let Q be a natural join query and $\mathcal{T}$ a decomposition of Q.*

(i) For each database $\mathbf{D}$, each cover of the query result $Q(\mathbf{D})$ over $\mathcal{T}$ has size $\mathcal{O}(|\mathbf{D}|^{\textsf{fhtw}(\mathcal{T})})$.

(ii) There are arbitrarily large databases $\mathbf{D}$ such that each cover of the query result $Q(\mathbf{D})$ over $\mathcal{T}$ has size $\Omega(|\mathbf{D}|^{\textsf{fhtw}(\mathcal{T})})$.

The size gaps between query results and their covers can be arbitrarily large. For any join query Q and database $\mathbf{D}$, it holds that $|Q(\mathbf{D})|= \mathcal{O}(|\mathbf{D}|^{\rho^*(Q)})$ and there are arbitrarily large databases $\mathbf{D}$ for which $|Q(\mathbf{D})|= \Omega(|\mathbf{D}|^{\rho^*(Q)})$ [4]. For acyclic queries, the fractional edge cover number ρ^* can be as large as $|Q|$, while the fractional hypertree width is one. Section 4 shows that the same gap also holds for time complexity.

▶ **Example 15.** We continue Example 8. The decomposition $\mathcal{T}$ has width one, which is minimal. The covers over $\mathcal{T}$, such as K_1 and K_2, have sizes upper bounded by the input database size. The minimum size of a cover over $\mathcal{T}$ is the maximum size of a relation used in the query (assuming the relations are globally consistent). In contrast, there are arbitrarily large databases of size N for which the query result has size $\Omega(N^2)$.

Proposition 10 and Theorem 14 give alternative equivalent characterizations of the size of a cover of a query result. The former gives it as the size of a *minimal edge cover* of the hypergraph of the query result over the attribute sets given by the bags of a decomposition $\mathcal{T}$, while the latter states it using the fractional hypertree width of $\mathcal{T}$ or equivalently the *maximum fractional edge cover* number over all the bags of $\mathcal{T}$. Most notably, whereas the former is an *integral* number, the latter is a *fractional* number.

This size gap between query results and their covers is precisely the same as for query results and their factorized representations called d-representations [23]. In this sense, covers can be seen as relational encodings of factorized representations of query results. We can easily translate covers into factorized representations. Appendix A gives a brief introduction to d-representations and a translation example.

▶ **Proposition 16.** *Given $(Q, \mathcal{T}, \mathbf{D})$, each cover K of the query result $Q(\mathbf{D})$ over $\mathcal{T}$ can be translated into a d-representation of $Q(\mathbf{D})$ of size $\mathcal{O}(|K|)$ and in time $\widetilde{\mathcal{O}}(|K|)$.*

The above translation allows us to extend the applicability of covers to known workloads over factorized representations, such as in-database optimization problems [2] and in particular learning regression models [22]. Nevertheless, it is practically desirable to process such workloads directly on covers, since this would avoid the indirection via factorized representations that comes with extra space cost and non-relational data representation. Aggregates, which are at the core of such workloads, can be computed directly on covers by joint scans of the projections of the cover onto the bags of the decomposition; alternatively, they can be computed by expressing any cover as the natural join of its bag projections and then pushing the aggregates past the join.

▶ **Example 17.** We consider the query $Q = R(A, B) \bowtie S(B, C)$ and its decomposition $\mathcal{T}$ with bags $\{A, B\}$ and $\{B, C\}$. To compute aggregates over the join result $Q(\mathbf{D})$, we can use any cover K of $Q(\mathbf{D})$ over $\mathcal{T}$. The expression for counting the number of result tuples is $\sum_{b\in\text{dom}(B)} \sum_{a\in\text{dom}(A)} \sum_{c\in\text{dom}(C)} \mathbf{1}_{R(a,b)} \cdot \mathbf{1}_{S(b,c)}$, where 1_E is the Kronecker delta that is evaluated to $\mathbf{1}$ if the event E is satisfied and $\mathbf{0}$ otherwise. We can compute it in one scan over K if K is sorted on (B, A, C) or (B, C, A). For each B-value b, we multiply the distinct numbers of A-values and of C-values paired with b in K, and we sum up these products over all B-values. We can rewrite this expression as follows: $\sum_{b\in\text{dom}(B)}(\sum_{a\in\text{dom}(A)} \mathbf{1}_{(a,b)\in\pi_{\{A,B\}}K})(\sum_{c\in\text{dom}(C)} \mathbf{1}_{(b,c)\in\pi_{\{B,C\}}K})$. This expression only uses the pairs (a, b) and (b, c) in K. The pairs (a, c), which make the difference among covers and are the culprits for the explosion in the size of the query result, are not needed.

Despite their succinctness over the explicit listing of tuples in a query result, any cover of the query result can be used to enumerate the result tuples with constant delay and extra space (data complexity) following linear-time pre-computation. In particular, the delay and the space are linear in the number of attributes of the query result which is as good as enumerating directly from the result. This complexity follows from Proposition 16 and the enumeration for factorized representations [23] with constant delay and extra space.

▶ **Corollary 18** (Proposition 16, Theorem 4.11 [23]). *Given $(Q, \mathcal{T}, \mathbf{D})$, the tuples in the query result $Q(\mathbf{D})$ can be enumerated from any cover K of $Q(\mathbf{D})$ over $\mathcal{T}$ with $\widetilde{\mathcal{O}}(|K|)$ pre-computation time and $\mathcal{O}(1)$ delay and extra space.*

An alternative way to achieve constant-delay enumeration with $\widetilde{\mathcal{O}}(|K|)$ pre-computation is by noting that the acyclic join queries considered in this paper are free-connex and thus allow for enumeration with constant delay and $\widetilde{\mathcal{O}}(|\mathbf{D}|)$ pre-computation [5]. An acyclic conjunctive query is called free-connex if its extension by a new relation symbol covering all attributes of the result remains acyclic [25]. Moreover, given a cover K over a decomposition $\mathcal{T}$, the natural join of the projections of K onto the bags of $\mathcal{T}$ is an acyclic query that computes the original query result (Proposition 6).

4 Computing Covers for Join Queries using Cover-Join Plans

Given an arbitrary join query and database, we can compute covers using a *monolithic* algorithm akin to known algorithms for computing factorized representations of query results [22]. However, is it possible to compute covers in a *compositional* way, by computing covers for one join at a time? In this section, we answer this question in the affirmative for acyclic natural join queries Q and globally consistent databases $\mathbf{D}$ with respect to Q.

For a triple $(Q, \mathcal{J}, \mathbf{D})$, where Q is an acyclic natural join query, $\mathcal{J}$ is a join tree of Q, and $\mathbf{D}$ is a database globally consistent with respect to Q, we use so-called cover-join plans to compute covers of the query result $Q(\mathbf{D})$ over the decomposition corresponding to the join tree $\mathcal{J}$. Such plans follow the structure of the join tree $\mathcal{J}$ and use a new binary join operator called cover-join. The cover-join of two relations yields a cover of their natural join. This approach is in the spirit of standard relational query evaluation. It is compositional in the sense that to compute a cover of the query result, it suffices to repeatedly compute a cover of the join of two relations. This is practical since it can be supported by existing query engines extended with the cover-join operator. We also show that, due to the binary nature of the cover-join operator, the cover-join plans cannot recover all possible covers of the query result. Furthermore, different plans may lead to different covers. Plans that do not follow the structure of a join tree may be unsound as they do not necessarily construct covers.

To compute covers for an arbitrary join query and database, we proceed in two stages. We first materialize the bags of a decomposition of the query so as to reduce it to an acyclic query Q over an extended database $\mathbf{D}$ that is now globally consistent with respect to Q (Proposition 3). We then use a cover-join plan to compute covers of $Q(\mathbf{D})$. The first step has a non-trivial time complexity overhead, whereas the second step is linearithmic. Overall, this strategy is worst-case optimal for computing covers for arbitrary join queries and databases.

4.1 The Cover-Join Operator

The building block of our approach to computing covers is the binary cover-join operator.

▶ **Definition 19** (Cover-Join). The cover-join of two relations R_1 and R_2, denoted by $R_1 \tilde{\bowtie} R_2$, computes a cover of their join result over the decomposition with bags $\mathcal{S}(R_1)$ and $\mathcal{S}(R_2)$.

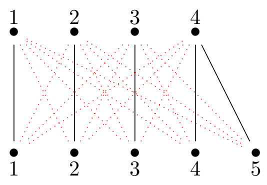
Following the alternative characterization of covers of a query result by minimal edge covers in the hypergraph of the query result (Proposition 10), the cover-join defines the relation $\mathit{rel}(M)$ of a minimal edge cover M of the hypergraph H of the result of the join $R_1 \bowtie R_2$ over the attribute sets $\mathcal{S}(R_1)$ and $\mathcal{S}(R_2)$. The hypergraph H is bipartite and consists of disjoint complete bipartite subgraphs. Since a cover is a minimal edge cover, it corresponds to a bipartite subgraph with the same number of nodes but a subset of the edges, where all paths can only have one or two edges. A cover cannot have unconnected nodes, since it would not be an edge cover. A path of three (or more) edges violates the minimality of the edge cover: Such a path $a_1 - b_1 - a_2 - b_2$ in a bipartite graph covers the four nodes, yet a minimal cover would only have the two edges $a_1 - b_1$ and $a_2 - b_2$.

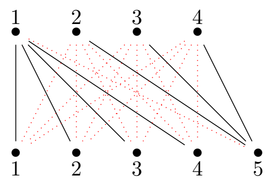
We can compute a cover of a join of two relations R_1 and R_2 in time $\widetilde{\mathcal{O}}(|R_1| + |R_2|)$, since it amounts to computing a minimal edge cover in a collection of disjoint complete bipartite graphs that encode the join result. The smallest size of a cover is given by the edge cover number of the bipartite graph representing the join result, which is the maximum of the sizes of the two sets of nodes in the graph [19]. The largest size can be achieved in case one of the two node sets has size one, in which case this is paired with all nodes in the second set. In case both sets have more than one node, the largest size is achieved when we pair one node from one of the two node sets with all but one node in the second set and then the remaining node in the second set with all but the already used node in the first set.

For the analysis in this paper, we assume that our cover-join algorithm may return any cover of the natural join of two relations. In practice, however, it makes sense to compute a cover of minimum size. We choose this cover as follows: For each complete bipartite hypergraph in the join result with node sets V_1 and V_2 such that $|V_1| \leq |V_2|$, we choose a minimum edge cover by pairing each node in V_1 with one distinct node in V_2 and all remaining nodes in V_2 with one node in V_1.

▶ **Proposition 20.** *Given two consistent relations R_1 and R_2, the cover-join computes a cover K of their join result over the decomposition with bags $\mathcal{S}(R_1)$ and $\mathcal{S}(R_2)$ in time $\widetilde{\mathcal{O}}(|R_1| + |R_2|)$ and with size* $\max\{|R_1|, |R_2|\} \leq |K| \leq |R_1| + |R_2|$.

▶ **Example 21.** Consider again the product $R_1(A) \bowtie R_2(B)$ in Example 9, where $R_1 = [2]$ and $R_2 = [n]$ with $n > 1$. Examples of covers of size n over the decomposition $\mathcal{T}$ with bags $\{A\}$ and $\{B\}$ are: $\{(1, i) \mid i \in [n] - \{k\}\} \cup \{(2, k)\}$ for any $k \in [n]$; $\{(1, i) \mid i \in [k]\} \cup \{(2, j + k) \mid j \in [n - k]\}$ for any $k \in [n - 1]$. If $R_1 = [m]$ with $m > n$, then examples of covers over $\mathcal{T}$ of minimum size m are: $\{(i, i) \mid i \in [k - 1]\} \cup \{(k - 1 + i, k + i) \mid i \in [n - k]\} \cup \{(n - 1 + i, k) \mid i \in [m - n + 1]\}$ for any $k \in [n]$. A cover over $\mathcal{T}$ of maximal size $n + m - 2$ is: $\{(1, i) \mid i \in [n - 1]\} \cup \{(j + 1, n) \mid j \in [m - 1]\}$. Below are depictions of the complete bipartite graph corresponding to the query result for $n = 4$ and $m = 5$, where the edges in a minimal edge cover are solid lines and all other edges are dotted. The left minimal edge cover corresponds to a cover over $\mathcal{T}$ of minimum size $m = 5$, while the right minimal edge cover corresponds to a cover over $\mathcal{T}$ of maximum size $n + m - 2 = 7$.





4.2 Cover-join Plans

We now compose cover-join operators into so-called cover-join plans to compute covers for acyclic natural join queries. Before we define such plans, we need to introduce some notation.

For a join tree $\mathcal{J}$ of a query Q, we write $\mathcal{J} = \mathcal{J}_1 \circ \mathcal{J}_2$ if $\mathcal{J}$ can be split into two non-empty subtrees $\mathcal{J}_1$ and $\mathcal{J}_2$ that are connected by a single edge in $\mathcal{J}$. Any subtree $\mathcal{J}'$ of $\mathcal{J}$ defines the subquery of Q that is the natural join of all relation symbols that are nodes in $\mathcal{J}'$.

▶ **Definition 22** (Cover-Join Plan)**.** Given $(Q, \mathcal{J}, \mathbf{D})$, a *cover-join plan φ over the join tree $\mathcal{J}$* is defined recursively as follows:

- If $\mathcal{J}$ consists of one node R, then $\varphi = R$. The plan φ returns R.
- If $\mathcal{J} = \mathcal{J}_1 \circ \mathcal{J}_2$ and φ_i is a cover-join plan over $\mathcal{J}_i$, then $\varphi = \varphi_1 \bowtie\!\!\!\!\diagdown\, \varphi_2$. The plan φ returns the result of $R_1 \bowtie\!\!\!\!\diagdown\, R_2$, where the relation R_i is returned by the plan φ_i ($i \in [2]$).

Lemma 23 states next that a cover-join plan computes a cover of the query result over the decomposition corresponding to a given join tree of the query.

▶ **Lemma 23.** *Given $(Q, \mathcal{J}, \mathbf{D})$ where $\mathbf{D} = \{R_i\}_{i \in [n]}$ is globally consistent with respect to Q, each cover-join plan over the join tree $\mathcal{J}$ computes a cover K of $Q(\mathbf{D})$ over the decomposition corresponding to $\mathcal{J}$ in time $\widetilde{\mathcal{O}}(|K|)$ and with size $\max_{i \in [n]}\{|R_i|\} \leq |K| \leq \sum_{i \in [n]} |R_i|$.*

Lemma 23 states three remarkable properties of cover-join plans. First, they compute covers compositionally: To obtain a cover of the entire query result it is sufficient to compute covers of the results for subqueries. More precisely, for a cover-join plan $\varphi_1 \bowtie\!\!\!\!\diagdown\, \varphi_2$, the sub-plans φ_1 and φ_2 compute covers for the subqueries defined by the joins of the relations in the join trees $\mathcal{J}_1$ and respectively $\mathcal{J}_2$. Then, the plan $\varphi_1 \bowtie\!\!\!\!\diagdown\, \varphi_2$ computes a cover for the join of the relations in the join tree $\mathcal{J} = \mathcal{J}_1 \circ \mathcal{J}_2$. Second, the output of a cover-join plan is always a cover, regardless which cover is picked at each cover-join operator in the plan. Third, it does not matter which cover-join plan we choose for a given join tree, the resulting covers are computed with the same time guarantee. Nevertheless, different plans for the same join tree may lead to different covers (Example 28).

These properties rely on the global consistency of the database and on the fact that the plans follow the structure of the join tree. For arbitrary databases, a cover-join operator may wrongly construct covers using dangling tuples at the expense of relevant tuples that are not anymore covered and therefore lost. Furthermore, plans that do not follow the structure of a join tree may be unsound (Example 26). Although each cover-join operator computes a cover of minimum size for the join of its input relations, the overall cover computed by a cover-join plan may not be a cover of minimum size of the query result (Example 35 in Appendix B).

▶ **Example 24.** A join tree that admits several splits can define many plans. For instance, the join tree for the query $R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(C, D)$ is the path $R_1 - R_2 - R_3$ and admits two possible splits that lead to the plans $\varphi_1 = (R_1(A, B) \bowtie\!\!\!\!\diagdown\, R_2(B, C)) \bowtie\!\!\!\!\diagdown\, R_3(C, D)$ and $\varphi_2 = R_1(A, B) \bowtie\!\!\!\!\diagdown\, (R_2(B, C) \bowtie\!\!\!\!\diagdown\, R_3(C, D))$. The relations are those in Figure 1, now calibrated. For this database, the covers computed by the sub-plans $R_1(A, B) \bowtie\!\!\!\!\diagdown\, R_2(B, C)$ and $R_2(B, C) \bowtie\!\!\!\!\diagdown\, R_3(C, D)$ correspond to full join results, since all join values only occur once in the relations. By taking any possible cover at each cover-join operator in the plans, both plans yield the same four possible covers of the query result: One of them is $\mathit{rel}(M)$ in Figure 1 and two of them are K_1 and K_2 in Example 8. The last cover is not depicted: It is the same as K_1 with the change that the values d_1 and d_2 are swapped between the first two rows.

A corollary of Proposition 3 and Lemma 23 is that covers over decompositions of *arbitrary* natural join queries can be computed in time proportional to their sizes.

▶ **Theorem 25** (Proposition 3, Lemma 23). *Given a natural join query Q, decomposition $\mathcal{T}$ of Q, and database $\mathbf{D}$, a cover of the query result $Q(\mathbf{D})$ over the decomposition $\mathcal{T}$ and with size $\mathcal{O}(|\mathbf{D}|^{\mathsf{fhtw}(\mathcal{T})})$ can be computed in time $\widetilde{\mathcal{O}}(|\mathbf{D}|^{\mathsf{fhtw}(\mathcal{T})})$.*

Given $(Q, \mathcal{T}, \mathbf{D})$ where Q is an arbitrary natural join query and $\mathbf{D}$ is an arbitrary database, we can compute a cover in four steps: construct $(Q', \mathcal{T}, \mathbf{D}')$ such that Q' is an acyclic natural join query, $\mathcal{T}$ corresponds to a join tree of Q' and $\mathbf{D}'$ consists of materializations of the bags of $\mathcal{T}$; turn $\mathbf{D}'$ into a globally consistent database $\mathbf{D}''$ with respect to Q'; turn $\mathcal{T}$ into a join tree $\mathcal{J}$ of Q' by replacing each bag by the corresponding relation symbol in Q'; and execute on $\mathbf{D}''$ a cover-join plan for Q' over $\mathcal{J}$. Since there are arbitrarily large databases for which the size bounds on covers are tight (Theorem 14), the cover-join plans, together with a worst-case optimal algorithm for materializing bags [21], represent a worst-case optimal algorithm for computing covers.

We conclude this section with three insights into the ability of cover-join plans to compute covers. We give an example of an unsound cover-join plan that does not follow the structure of a join tree. We then note the incompleteness of our cover-join plans due to the binary nature of the cover-join operator. We give an example of a cover that cannot be computed with our cover-join plans, but can be computed using a multi-way cover-join operator. Finally, we give an example showing that distinct cover-join plans over the same (or also distinct) join trees can yield incomparable sets of covers.

▶ **Example 26** (Unsound plan). Consider the query $Q = R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(C, D)$, the following database with relations R_1, R_2, and R_3, and four relations computed by cover-joining two of the three relations:

R_1	
A	B
a	b_1
a	b_2

R_2	
B	C
b_1	c_1
b_2	c_2

R_3	
C	D
c_1	d
c_2	d

$K_{1,3}$			
A	B	C	D
a	b_1	c_1	d
a	b_2	c_2	d

$K'_{1,3}$			
A	B	C	D
a	b_1	c_2	d
a	b_2	c_1	d

$K_{1,2}$		
A	B	C
a	b_1	c_1
a	b_2	c_2

$K_{2,3}$		
B	C	D
b_1	c_1	d
b_2	c_2	d

Following Definition 22, the plan $(R_1(A, B) \overset{\circ}{\bowtie} R_3(C, D)) \overset{\circ}{\bowtie} R_2(B, C)$ would require a split $\mathcal{J}_{1,3} \circ \mathcal{J}_2$ of a join tree, where the join tree $\mathcal{J}_{1,3}$ has two nodes R_1 and R_3 while the join tree $\mathcal{J}_2$ has one node R_2. However, there is no join tree that allows such a split.

The cover-join $R_1(A, B) \overset{\circ}{\bowtie} R_3(C, D)$ computes one of the two covers $K_{1,3}$ and $K'_{1,3}$. The result of the join of $K'_{1,3}$ and R_2 is empty and so is the cover-join. This means that this plan does not always compute a cover, which makes it unsound.

This problem cannot occur with cover-join plans over join trees of Q. The only cover-join plans over join trees of Q are (up to commutativity) $(R_1(A, B) \overset{\circ}{\bowtie} R_2(B, C)) \overset{\circ}{\bowtie} R_3(C, D)$ and $R_1(A, B) \overset{\circ}{\bowtie} (R_2(B, C) \overset{\circ}{\bowtie} R_3(C, D))$. The only cover of $R_1(A, B) \overset{\circ}{\bowtie} R_2(B, C)$ is $K_{1,2}$ above, which can be cover-joined with R_3. The only cover of $R_2(B, C) \overset{\circ}{\bowtie} R_3(C, D)$ is $K_{2,3}$ above, which can be cover-joined with R_1.

▶ **Example 27** (Cover-Join Incompleteness). Consider the product query $Q = R_1(A) \bowtie R_2(B) \bowtie R_3(C)$, the following database $\mathbf{D}$ with relations R_1, R_2, and R_3 and one cover K of the query result over the decomposition with bags $\{A\}$, $\{B\}$, and $\{C\}$:

R_1
A
a_1
a_2

R_2
B
b_1
b_2

R_3
C
c_1
c_2

K		
A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_1	c_2

A decomposition of Q can have up to three bags which are not included in other bags.

In case of decompositions with three bags, each bag consists of exactly one attribute. These decompositions correspond to the join trees that are permutations of the three relation symbols. There are three possible cover-join plans (up to commutativity) over these join trees: $\varphi_1 = R_1(A)\bowtie(R_2(B)\bowtie R_3(C))$, $\varphi_2 = R_2(B)\bowtie(R_1(A)\bowtie R_3(C))$ and $\varphi_3 = R_3(C)\bowtie(R_1(A)\bowtie R_2(B))$. None of these plans can yield the cover K above. As discussed after Definition 19, a minimal edge cover corresponding to a cover computed by a *binary* cover-join operator can only have paths of one or two edges. For instance, $\pi_{\{A,B\}}K$, which should correspond to a cover of $R_1(A)\bowtie R_2(B)$, has the path of three edges $b_2 - a_1 - b_1 - a_2$. The cover-join $R_1(A)\bowtie R_2(B)$ would not create this path since it corresponds to a non-minimal edge cover. Similarly, $\pi_{\{A,C\}}K$ and $\pi_{\{B,C\}}K$ have paths of three edges.

For decompositions with two bags, two of the three attributes are in the same bag. Without loss of generality, assume A and B are in the same bag. Following Proposition 3, this bag is covered by a new relation $R_{1,2}$ that is the product of R_1 and R_2. This means that K has to be the cover of $R_{1,2}(A,B)\bowtie R_3(C)$, yet $\pi_{\{A,B\}}K$ is not $R_{1,2}$!

The decomposition with one bag consisting of all three attributes has this bag covered by a new relation that is the product of the three relations. This relation is the Cartesian product of the three relations that is the full query result and different from $K = \pi_{\{A,B,C\}}K$.

We conclude that K cannot be computed using (binary) cover-join plans.

▶ **Example 28** (Incomparable Sets of Covers). Consider the product query $Q = R_1(A) \bowtie R_2(B) \bowtie R_3(C)$ and the following database $\{R_1, R_2, R_3\}$:

R_1
A
a_1
a_2

R_2
B
b_1
b_2

R_3
C
c_1
c_2
c_3

K		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_1	b_2	c_3

$K_{1,2}$	
A	B
a_1	b_1
a_2	b_2

$K'_{1,2}$	
A	B
a_1	b_2
a_2	b_1

Let us consider the join tree $\mathcal{J} = R_1 - R_2 - R_3$ of Q. There are (up to commutativity) two possible cover-join plans over $\mathcal{J}$: $\varphi_1 = R_1(A)\bowtie(R_2(B)\bowtie R_3(C))$ and $\varphi_2 = (R_1(A)\bowtie R_2(B))\bowtie R_3(C)$. The above relation K is a cover of the result of Q and can be computed by φ_1, which cover-joins $R_1(A)$ and a cover of the join of $R_2(B)$ and $R_3(C)$. This cover cannot be computed by φ_2. Indeed, φ_2 first cover-joins $R_1(A)$ and $R_2(B)$, yielding $K_{1,2}$ or $K'_{1,2}$ as the only possible covers. Then, cover-joining any of them with $R_3(C)$ does not yield the cover K since $\pi_{\{A,B\}}K$ is different from both $K_{1,2}$ and $K'_{1,2}$. Similarly, φ_2 computes covers that cannot be computed by φ_1.

5 Covers for Functional Aggregate Queries

We first give a brief introduction to functional aggregate queries (FAQ) [18]. A detailed description can be found in the extended report [17].

Given an attribute set S, we use a_S to indicate that tuple a has schema S. For $S' \subseteq S$, we denote by $\mathsf{a}_{S'}$ the restriction of a to S'. A functional aggregate query has the following form (slightly adapted to our notation):

$$\varphi(\mathsf{a}_{\{A_1,\ldots,A_f\}}) = \bigoplus_{a_{f+1}\in\mathsf{dom}(A_{f+1})}^{(f+1)} \cdots \bigoplus_{a_n\in\mathsf{dom}(A_n)}^{(n)} \bigotimes_{S\in\mathcal{E}} \psi_S(\mathsf{a}_S), \text{ where:} \qquad (1)$$

- $H = (\mathcal{V}, \mathcal{E})$ is the multi-hypergraph of the query with $\mathcal{V} = \{A_i\}_{i\in[n]}$.
- Dom is a fixed (output) domain, such as $\{\mathsf{true},\mathsf{false}\}$, $\{0,1\}$, or $\mathbb{R}^+$.
- $\mathcal{V}_{\text{free}} = \{A_1,\ldots,A_f\}$ is the set of result or free attributes; all other attributes are bound.
- For each attribute A_i with $i > f$, $\oplus^{(i)}$ is a binary (aggregate) operator on the domain Dom. Different bound attributes may have different aggregate operators.
- For each attribute A_i with $i > f$, either $\oplus^{(i)}$ is $\otimes$ or $(\mathsf{Dom}, \oplus^{(i)}, \otimes)$ forms a commutative semiring with the same additive identity $\mathbf{0}$ and multiplicative identity $\mathbf{1}$ for all semirings.
- For every hyperedge S in $\mathcal{E}$, $\psi_S : \prod_{A\in S} \mathsf{dom}(A) \to \mathsf{Dom}$ is an (input) function.

FAQs are a semiring generalization of aggregates over join queries, where the aggregates are the operators $\oplus^{(i)}$ and the natural join is expressed by $\bigotimes_{S\in\mathcal{E}} \psi_S(\mathsf{a}_S)$. The listing representation R_{ψ_S} of a function ψ_S is a relation over the schema $S \cup \{\psi_S(S)\}$ which consists of all input-output pairs for ψ_S where the output is non-zero, i.e., R_{ψ_S} contains a tuple $\mathsf{a}_{S\cup\{\psi_S(S)\}}$ if and only if $\psi_S(\mathsf{a}_S) = \mathsf{a}_{\psi_S(S)} \neq \mathbf{0}$. An input database for φ contains for each ψ_S its listing representation. We say that $\mathcal{T}$ is a decomposition of φ if $\mathcal{T}$ is a decomposition of the hypergraph H of φ. Given an FAQ φ and database $\mathbf{D}$, the FAQ-problem is to compute the query result $\varphi(\mathbf{D})$.

Each FAQ φ has an FAQ-width $\mathsf{faqw}(\varphi)$ which is defined similarly to the fractional hypertree width of the hypergraph of φ. For instance, in case where all attributes of φ are free, $\mathsf{faqw}(\varphi)$ is equal to the fractional hypertree width of the hypergraph of φ.

Given an FAQ φ and a database $\mathbf{D}$, the $\mathsf{InsideOut}$ algorithm [18] solves the FAQ-problem as follows. First, it eliminates all bound attributes along with their corresponding aggregate operators by performing equivalence-preserving transformations on φ. Then, it computes the listing representation of the remaining query. The algorithm runs in time $\widetilde{\mathcal{O}}(|\mathbf{D}|^{\mathsf{faqw}(\varphi)} + Z)$ where Z is the size of the output, i.e., the listing representation of φ.

We can compute a cover of the result of a given FAQ φ in time $\widetilde{\mathcal{O}}(|\mathbf{D}|^{\mathsf{faqw}(\varphi)})$, which does not depend on the size of the listing representation of φ. Our strategy is as follows. We first eliminate all bound attributes in φ by using $\mathsf{InsideOut}$ resulting in an FAQ φ'. We then take a decomposition $\mathcal{T}$ of φ' and compute bag functions β_B, $B \in \mathcal{S}(\mathcal{T})$, with $\varphi'(\mathsf{a}_{\mathcal{V}_{\text{free}}}) = \bigotimes_{B\in\mathcal{S}(\mathcal{T})} \beta_B(\mathsf{a}_B)$. Finally, we compute a cover of the join result of the listing representations of the bag functions over the extension of $\mathcal{T}$ that contains, for each bag B, the attribute $\beta_B(B)$ for the values of the function β_B. Keeping the $\beta_B(B)$-values of the bag functions in the cover is necessary for recovering the output values of φ when enumerating the result of φ from the cover.

► **Example 29.** We consider the following FAQ φ over the sum-product semiring $(\mathbb{N}, +, \cdot)$ (for simplicity we skip the explicit iteration over the domains of the attributes in φ):

$$\varphi(a,b,d) = \sum_{c,e,f,g,h} \psi_1(a,b,c) \cdot \psi_2(b,d,e) \cdot \psi_3(d,e,f) \cdot \psi_4(f,h) \cdot \psi_5(e,g), \text{ where}$$

φ, ψ_1, ψ_2, ψ_3, ψ_4 and ψ_5 are over $\{A,B,D\}$, $\{A,B,C\}$, $\{B,D,E\}$, $\{D,E,F\}$, $\{F,H\}$ and $\{E,G\}$, respectively. We first run InsideOut on φ to eliminate the bound attributes and obtain the following FAQ:

$$\varphi'(a,b,d) = \underbrace{\Big(\sum_c \psi_1(a,b,c)\Big)}_{\psi_6(a,b)} \cdot \underbrace{\sum_e \Big(\psi_2(b,d,e) \cdot \underbrace{\sum_f \Big(\psi_3(d,e,f) \cdot \underbrace{\sum_h \psi_4(f,h)}_{\psi_7(f)}\Big)}_{\psi_9(d,e)} \cdot \underbrace{\sum_g \psi_5(e,g)}_{\psi_8(e)}\Big)}_{\psi_{10}(b,d)}.$$

We consider the decomposition $\mathcal{T}$ of φ' with two bags $B_1 = \{A,B\}$ and $B_2 = \{B,D\}$ and bag functions ψ_6 and respectively ψ_{10}. Then, we execute the cover-join plan $R_{\psi_6} \bowtie R_{\psi_{10}}$ over the extended decomposition $\mathcal{T}'$ with bags $\{A,B,\psi_6(A,B)\}$ and $\{B,D,\psi_{10}(B,D)\}$. While the computation of the result of φ' can take quadratic time, the above cover-join plan takes linear time. We exemplify the computation of the cover-join plan. Assume the following tuples in ψ_6 and ψ_{10}, where $\gamma_1,\ldots,\gamma_4,\delta_1,\ldots,\delta_3 \in \mathbb{N}$:

ψ_6

A	B	$\psi_6(A,B)$
a_1	b_1	γ_1
a_2	b_1	γ_2
a_3	b_2	γ_3
a_4	b_2	γ_4

ψ_{10}

B	D	$\psi_{10}(B,D)$
b_1	d_1	δ_1
b_1	d_2	δ_2
b_2	d_3	δ_3

K

A	B	D	$\psi_6(A,B)$	$\psi_{10}(B,D)$
a_1	b_1	d_1	γ_1	δ_1
a_2	b_1	d_2	γ_2	δ_2
a_3	b_2	d_3	γ_3	δ_3
a_4	b_2	d_3	γ_4	δ_3

The relation K is a possible cover computed by the cover-join plan. The cover carries over the aggregates in columns $\psi_6(A,B)$ and $\psi_{10}(B,D)$, one per bag of $\mathcal{T}'$. The aggregate of the first tuple in K is $\gamma_1 \cdot \delta_1$ (or $\gamma_1 \otimes \delta_1$ under a semiring with multiplication $\otimes$).

The following theorem relies on Lemma 23 and Theorem 25 that give an upper bound on the time complexity for constructing covers of join results.

▶ **Theorem 30.** *For each FAQ φ and database $\mathbf{D}$, a cover of the query result $\varphi(\mathbf{D})$ can be computed in time $\widetilde{\mathcal{O}}(|\mathbf{D}|^{\mathsf{faqw}(\varphi)})$.*

Any enumeration algorithm for covers of join results can be used to enumerate the tuples of an FAQ result from one of its covers. We thus have the following corollary:

▶ **Corollary 31** (Corollary 18)**.** *Given a cover K of the result $\varphi(\mathbf{D})$ of an FAQ φ over a database $\mathbf{D}$, the tuples in the query result $\varphi(\mathbf{D})$ can be enumerated with $\widetilde{\mathcal{O}}(|K|)$ pre-computation time and $\mathcal{O}(1)$ delay and extra space.*

6 Conclusion

Results of join and functional aggregate queries entail redundancy in both their computation and representation. In this paper we propose the notion of covers of query results to reduce such redundancy. While covers can be more succinct than the query results, they nevertheless enjoy desirable properties such as listing representation and constant-delay enumeration of result tuples. For a given database and a join or functional aggregate query, the query result can be normalized as a globally consistent database over an acyclic schema. Covers represent one-relational, lossless, linear-size encodings of such normalized databases.

▶ **Definition 32.** **borged** `/bôrjd/` : Buy One Relation, Get Entire Database!

References

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

2 Mahmoud Abo-Khamis, Hung Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. In-database learning with sparse tensors. In *PODS*, 2018. To appear.

3 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

4 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013.

5 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *CSL*, pages 208–222, 2007.

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

7 Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

8 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

9 Hugh Darwen, C. J. Date, and Ronald Fagin. A normal form for preventing redundant tuples in relational databases. In *ICDT*, pages 114–126, 2012.

10 Ronald Fagin. Normal forms and relational database operators. In *SIGMOD*, pages 153–160, 1979.

11 Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: Getting to the core. *ACM Trans. Datab. Syst.*, 30(1):174–210, 2005.

12 Georg Gottlob. Computing cores for data exchange: New algorithms and practical solutions. In *PODS*, pages 148–159, 2005.

13 Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. Generalized hypertree decompositions: NP-hardness and tractable variants. *J. ACM*, 56(6):30:1–30:32, 2009.

14 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Alg.*, 11(1):4, 2014.

15 Claudio Gutierrez, Carlos Hurtado, and Alberto O. Mendelzon. Foundations of semantic web databases. In *PODS*, pages 95–106, 2004.

16 Pavol Hell and Jaroslav Nešetřil. The core of a graph. *Discrete Mathematics*, 109(1):117–126, 1992.

17 Ahmet Kara and Dan Olteanu. Covers of query results. *CoRR*, abs/1709.01600, 2017. URL: `http://arxiv.org/abs/1709.01600`.

18 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. In *PODS*, pages 13–28, 2016.

19 E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover Publications, 2001.

20 Dániel Marx. Approximating fractional hypertree width. *ACM Trans. Alg.*, 6(2):29:1–29:17, 2010.

21 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.

22 Dan Olteanu and Maximilian Schleich. Factorized Databases. *SIGMOD Record*, 45(2):5–16, 2016.

23 Dan Olteanu and Jakub Závodnỳ. Size bounds for factorised representations of query results. *ACM Trans. Datab. Syst.*, 40(1):2:1–2:44, 2015.

24 Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning Linear Regression Models over Factorized Joins. In *SIGMOD*, pages 3–18, 2016.

25 Luc Segoufin. Constant delay enumeration for conjunctive queries. *SIGMOD Record*, 44(1):10–17, 2015.

26 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.

A From Covers to D-Representations

We next give a brief introduction to d-representations; for a detailed description, we refer the reader to the literature [23]. We then discuss a translation from covers to d-representations.

A.1 D-Representations in a Nutshell

D-representations are a lossless succinct representation for relational data. A d-representation is a set of relational algebra expressions $\{N_1 := E_1, \ldots, N_n := E_n\}$, where each N_i is a unique name and each E_i is a relational algebra expression with unions, Cartesian products, singleton relations, i.e., unary relations with one tuple, and name references in place of singleton relations. The size $|E|$ of a d-representation E is the number of its singletons.

We consider a special class of d-representations that encode results of join queries and whose nesting structure is given by so-called d-trees. In the literature, d-trees are defined as orderings on query variables. We give here an alternative, equivalent definition that is in line with our notion of fractional hypertree decomposition. Given a query Q, a d-tree of Q is a decomposition of Q where each bag is partitioned into one attribute A, called the *bag attribute*, and a set of attributes, called the *key* of A and denoted by *key*(A). There is one bag per distinct attribute A in Q. Each decomposition $\mathcal{T}$ of a query Q can be translated into a d-tree $\mathcal{T}'$ of Q with $\mathsf{fhtw}(\mathcal{T}') \leq \mathsf{fhtw}(\mathcal{T})$ (Proposition 9.3 in [23]). Given a query Q, a d-tree $\mathcal{T}$ of Q, and a database $\mathbf{D}$, a d-representation E of $Q(\mathbf{D})$ over $\mathcal{T}$ with size $\mathcal{O}(|\mathbf{D}|^{\mathsf{fhtw}(\mathcal{T})})$ can be computed in time $\widetilde{\mathcal{O}}(|\mathbf{D}|^{\mathsf{fhtw}(\mathcal{T})})$ (Theorem 7.13 and Proposition 8.2 in [23]).

▶ **Example 33.** We consider the path query $Q = R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,D)$. Figure 2 depicts a database with relations R_1, R_2 and R_3 and the result of Q over the input database $\{R_1, R_2, R_3\}$. It also shows a decomposition $\mathcal{T}$ of Q and a cover K of the query result over $\mathcal{T}$. Finally, it depicts a d-tree $\mathcal{T}'$ (right below) derived from $\mathcal{T}$ by using the translation in the proof of Proposition 9.3 in [23].

D-representations can be encoded as parse graphs and sets of multi-maps. Figure 3 visualizes the two encodings for the d-representation of the query result from Figure 2 over the d-tree $\mathcal{T}'$. The parse graph follows the structure of the d-tree. At the top level we have a union of B-values. Then, given any B-value, the A-values are independent of the values for C and D. Therefore, under each B-value, the A-values are represented in a different branch than the values for C and D. Within the branches for C and D, the values are first grouped by C and then by D. The information on keys is used to share subtrees across branches. Since the key of attribute D is C, all C-nodes with the same value point to the same union of D-values. In our example, both c_1-nodes point to the same set $\{d_1, d_2\}$ of D-values.

The cover K from Figure 2 can be mapped immediately to the parse graph: Under each product node, we take a minimum number of combinations of its children to ensure that every value under the product node occurs in one of these combinations. To enumerate the tuples in the query result, it suffices to choose in turn one branch of each union node and all branches of each product node. For instance, the left product node represents the combinations of $\{a_1, a_2\}$ with $\{d_1, d_2\}$, together with the values b_1 and c_1. There are four combinations, so four tuples in the result. The first two tuples in the cover represent two of them, yet they are sufficient to recover all these tuples.

R_1

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_2

R_2

B	C
b_1	c_1
b_2	c_1

R_3

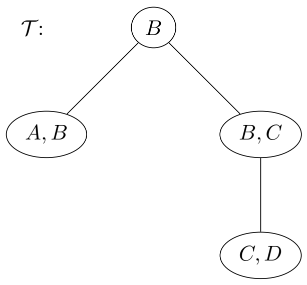
C	D
c_1	d_1
c_1	d_2

$Q(\mathbf{D})$

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_1	d_2
a_2	b_1	c_1	d_1
a_2	b_1	c_1	d_2
a_3	b_2	c_1	d_1
a_3	b_2	c_1	d_2
a_4	b_2	c_1	d_1
a_4	b_2	c_1	d_2

$K \subseteq Q(\mathbf{D})$

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_1	c_1	d_1
a_3	b_2	c_1	d_2
a_4	b_2	c_1	d_2



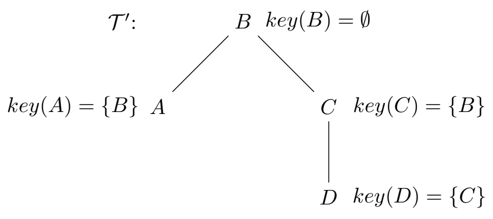
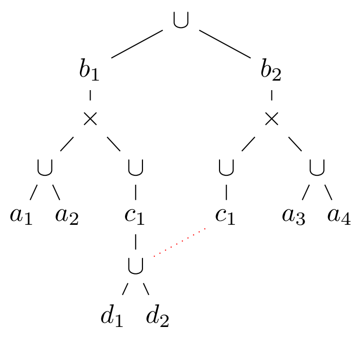


Figure 2 Top row: database $\mathbf{D} = \{R_1, R_2, R_3\}$, the result $Q(\mathbf{D})$ of the path query $Q = R_1 \bowtie R_2 \bowtie R_3$, and a cover $K \subseteq Q(\mathbf{D})$ over the decomposition $\mathcal{T}$; bottom row: decomposition $\mathcal{T}$ of Q and an equivalent d-tree $\mathcal{T}'$.



m_B

$key(B)$	B
()	b_1
()	b_2

m_A

$key(A)$	A
b_1	a_1
b_1	a_2
b_2	a_3
b_2	a_4

m_C

$key(C)$	C
b_1	c_1
b_2	c_1

m_D

$key(D)$	D
c_1	d_1
c_1	d_2

Figure 3 A d-representation encoded as a parse graph (left) and as a set of multimaps (right).

```
**cover2factorization** (cover $K$, decomposition $\mathcal{T}$)
convert $\mathcal{T}$ into an equivalent d-tree $\mathcal{T}'$ following Proposition 9.3 in [23];
let $\mathbf{V}$ be the set of attributes in $\mathcal{T}'$;
foreach attribute $A \in \mathbf{V}$ do
    create multi-map $m_A : \prod_{X \in key(A)} \text{dom}(X) \mapsto \text{dom}(A)$;
foreach tuple $t \in K$ do
    foreach attribute $A \in \mathbf{V}$ do
        insert assignment $\pi_{key(A)} t \mapsto \pi_A t$ into $m_A$;
return $\{m_A\}_{A \in \mathbf{V}}$;
```

Figure 4 Translating a cover K over a decomposition $\mathcal{T}$ into an equivalent d-representation.

The (multi-)map encoding of a d-representation consists of one map for each bag attribute: m_A maps tuples over the attributes in $key(A)$ to values of A. Figure 3 shows these maps as relations with columns for the key attributes (the map keys) and the column for the attribute A itself (the map payload). For instance, $m_A(b_1) = a_1$ and $m_A(b_1) = a_2$, while $m_C(b_1) = c_1$. Since $key(A) = \{B\}$ and there are two B-values in the d-representation leading to the sets $\{a_1, a_2\}$ and $\{a_3, a_4\}$, respectively, m_A maps the B-value b_1 to both A-values a_1 and a_2 and the B-value b_2 to both A-values a_3 and a_4.

A.2 Translating Covers into D-Representations

Figure 4 gives an algorithm that constructs an equivalent d-representation from a cover over a decomposition. Both the cover K and the output d-representation are for the same query result $Q(\mathbf{D})$ of a query Q. The decomposition $\mathcal{T}$ is for the query Q.

The algorithm creates a multi-map for each attribute A and populates it with assignments of tuples over the keys of A to the values of A as encountered in the tuples of the cover.

▶ **Example 34.** We consider the cover K over the decomposition $\mathcal{T}$ in Figure 2 and the d-tree $\mathcal{T}'$ equivalent to $\mathcal{T}$. The cover K is translated into a d-representation over $\mathcal{T}'$ as follows. On the first tuple (a_1, b_1, c_1, d_1), we add $() \mapsto b_1$ to m_B, $b_1 \mapsto a_1$ to m_A, $b_1 \mapsto c_1$ to m_C, and $c_1 \mapsto d_1$ to m_D, where $()$ means the empty tuple. On the second tuple (a_2, b_1, c_1, d_1), we only change m_A by adding $b_1 \mapsto a_2$ to m_A. On the third tuple (a_3, b_2, c_1, d_2), we add the following new assignments: $() \mapsto b_2$ to m_B, $b_2 \mapsto a_3$ to m_A, $b_2 \mapsto c_1$ to m_C, and $c_1 \mapsto d_2$ to m_D. On the last tuple (a_4, b_2, c_1, d_2), we add the new assignment $b_2 \mapsto a_4$ to m_A.

B Cover-Join Plans Computing Covers of Non-Minimum Size

▶ **Example 35.** We consider the acyclic natural join query $Q = R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(C, D)$, the database $\mathbf{D} = \{R_1, R_2, R_3\}$ globally consistent with respect to Q, and the join tree $\mathcal{J} = R_1 - R_2 - R_3$. The relations R_i are depicted below.

R_1

A	B
a_1	b_1
a_2	b_1
a_3	b_1

R_2

B	C
b_1	c_1
b_1	c_2

R_3

C	D
c_1	d_1
c_2	d_1
c_2	d_2

K

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_1	c_2	d_1
a_3	b_1	c_2	d_2

$K_{1,2}$

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_3	b_1	c_2

K'

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_1	c_1	d_1
a_3	b_1	c_2	d_1
a_3	b_1	c_2	d_2

The relation K is a cover of the query result $Q(\mathbf{D})$ over the decomposition $\mathcal{T}$ corresponding to $\mathcal{J}$. It follows from Proposition 13 that every cover of $Q(\mathbf{D})$ over $\mathcal{T}$ must have size at least three. Hence, K is a minimum-sized cover of $Q(\mathbf{D})$ over $\mathcal{T}$.

We take the cover-join plan $(R_1 \bowtie R_2) \bowtie R_3$ over $\mathcal{J}$ and assume that the cover-join operator computes for each two input relations R and R', a minimum-sized cover of $R \bowtie R'$ over the decomposition with bags $\mathcal{S}(R)$ and $\mathcal{S}(R')$. Then, a possible output of the sub-plan $R_1 \bowtie R_2$ is the relation $K_{1,2}$. A possible result of the cover-join of $K_{1,2}$ with R_3 is the relation K', which is a valid cover of $Q(\mathbf{D})$ over $\mathcal{T}$, but not a minimum-sized cover of $Q(\mathbf{D})$ over $\mathcal{T}$.

C Covers for Equi-Join Queries

In this section, we extend the class of queries from natural join queries to arbitrary equi-join queries, whose relation symbols may map to the same database relation.

Equi-join Queries. An equi-join query, aka full conjunctive query, has the form $Q = \sigma_\psi(R_1(S_1) \times \ldots \times R_n(S_n))$, where each R_i is a relation symbol with schema S_i and ψ is a conjunction of equalities of the form $A_1 = A_2$ with attributes A_1 and A_2. We require that all relation symbols in the query as well as all attributes occurring in the schemas of the relation symbols are distinct. We assume that each query comes with mappings $(\lambda, \{\mu_{R_i}\}_{i\in[n]})$, called the signature mappings of Q, where λ maps the relation symbols in Q to relation symbols in the schema of the database and each μ_{R_i} is a bijective mapping from the attributes of R_i to the attributes of $\lambda(R_i)$. Since we do not require λ to be injective, distinct relation symbols in Q might refer to the same relation in the database (cf. Example 36). The joins in equi-join queries are expressed by the equalities in ψ. The transitive closure ψ^+ of ψ under the equality on attributes defines the attribute equivalence classes: The equivalence class $\mathcal{A}$ of an attribute A is the set consisting of A and of all attributes equal to A in ψ^+. For a set S of attributes, S^+ denotes the set of attributes transitively equivalent to those in S.

The Hypergraph and the hypertree decompositions of Q are defined just like for natural join queries with the additional requirement that each hyperedge or bag is closed with respect to the equivalence classes in ψ^+. More formally, the hypergraph of Q consists of one node A for each attribute A in Q and one edge $\mathcal{S}(R)^+$ for each relation symbol $R \in \mathcal{S}(Q)$. Similarly, a hypertree decomposition $\mathcal{T}$ (of the hypergraph H) of Q is a pair (T, χ), where T is a tree and χ is a function mapping each node in T to a set V^+ where V is a subset of the nodes of H. All other notions and notations introduced in Section 2 as well as the definitions of result preservation and covers in Section 3 carry over to equi-join queries without any change.

▶ **Example 36.** We consider the equi-join query $Q = \sigma_\psi(R_1(A_1, A_2) \times R_2(A_3, A_4))$ where $\psi = \{A_2 = A_3\}$. Let $(\lambda, \{\mu_{R_1}, \mu_{R_2}\})$ be the signature mappings of Q. Assume that $\lambda(R_1) = \lambda(R_2) = R$, $\mu_{R_1}(A_1) = \mu_{R_2}(A_4) = A$ and $\mu_{R_1}(A_2) = \mu_{R_2}(A_3) = B$, i.e., both relation symbols are mapped to the same relation symbol R, attributes A_1 and A_4 are mapped to attribute A and attributes A_2 and A_3 are mapped to attribute B. Let $\mathbf{D} = \{R\}$ where R is defined as in Figure 5. The figure depicts in the top row the query result $Q(\mathbf{D})$, a cover K of the query result over the decomposition $\mathcal{T}$ depicted in the bottom row and two relations R_1', R_2' obtained from R by the application of Proposition 37 (given below). The bottom row shows the hypergraph of Q, the hypergraph H of $Q(\mathbf{D})$ over the attribute sets $\{\{A_1, A_2, A_3\}, \{A_2, A_3, A_4\}\}$, and a minimal edge cover M of H with $rel(M) = K$.

R

A	B
a_1	b_1
a_2	b_1
a_1	b_2
a_2	b_2

$Q(\mathbf{D})$

A_1	A_2	A_3	A_4
a_1	b_1	b_1	a_1
a_1	b_1	b_1	a_2
a_2	b_1	b_1	a_1
a_2	b_1	b_1	a_2
a_1	b_2	b_2	a_1
a_1	b_2	b_2	a_2
a_2	b_2	b_2	a_1
a_2	b_2	b_2	a_2

$K = rel(M)$

A_1	A_2	A_3	A_4
a_1	b_1	b_1	a_1
a_2	b_1	b_1	a_2
a_1	b_2	b_2	a_2
a_2	b_2	b_2	a_1

R'_1

A_1	A_2	A_3
a_1	b_1	b_1
a_2	b_1	b_1
a_1	b_2	b_2
a_2	b_2	b_2

R'_2

A_2	A_3	A_4
b_1	b_1	a_1
b_1	b_1	a_2
b_2	b_2	a_1
b_2	b_2	a_2

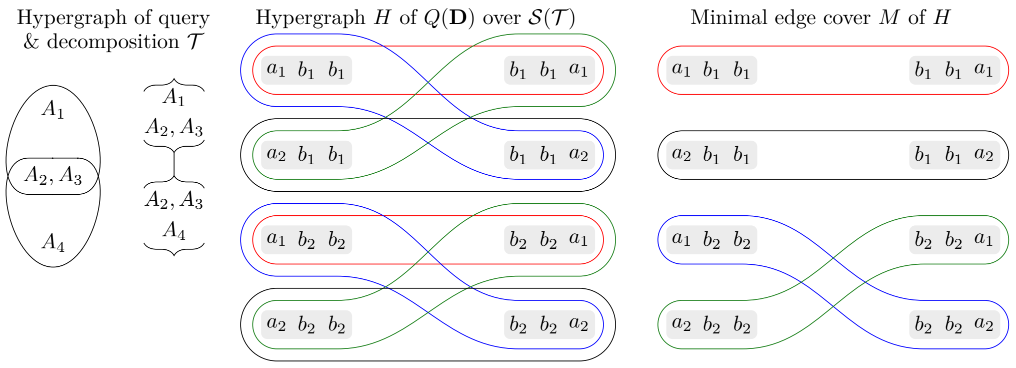


Figure 5 Top row: database $\mathbf{D} = \{R\}$, the result $Q(\mathbf{D})$ of the query Q in Example 36, a cover K of $Q(\mathbf{D})$ over $\mathcal{T}$, and relations R'_1, R'_2 obtained from R by the application of Proposition 37; bottom row: the hypergraph of Q, a decomposition $\mathcal{T}$ of Q, the hypergraph of $Q(\mathbf{D})$ over the attribute sets $\mathcal{S}(\mathcal{T})$, and a minimal edge cover M of this hypergraph.

Adaption of the results on covers to equi-join queries. Due to the following two propositions, all results on covers in Sections 3 and 4 carry over to equi-join queries.

▶ **Proposition 37.** *Given an equi-join query Q, a decomposition $\mathcal{T}$ of Q, and a database $\mathbf{D}$, there exist a natural join query Q' and a database $\mathbf{D}'$ such that: $Q'(\mathbf{D}') = Q(\mathbf{D})$, Q' has the decomposition $\mathcal{T}$ and can be constructed in time $\mathcal{O}(|Q|)$, and $\mathbf{D}'$ can be constructed in time $\mathcal{O}(|\mathbf{D}|)$.*

We briefly explain the construction. The query Q' is obtained from Q by replacing each relation symbol $R(S)$ in Q by a relation symbol $R'(S^+)$. The database $\mathbf{D}'$ contains, for each relation symbol $R'(S^+)$ in Q', a relation over the same schema that is obtained from relation $\lambda(R(S))$ as follows: for each attribute A contained in S^+ but not in S, $\lambda(R(S))$ is extended by a new A-column that is a copy of any B-column in $\lambda(R(S))$ such that A is equivalent to B. Figure 5 gives in the top row two relations R'_1 and R'_2 that result from relation R by the application of Proposition 37 in case Q is defined as in Example 36.

It follows from Proposition 37 that, since $Q'(\mathbf{D}') = Q(\mathbf{D})$, any relation K is a cover of $Q(\mathbf{D})$ over $\mathcal{T}$ if and only if K is a cover of $Q'(\mathbf{D}')$ over $\mathcal{T}$. Given the construction times for Q' and D', all our results on natural join queries in Sections 3 and 4, except the lower size bound on covers in Theorem 14(ii), hold for equi-join queries, too.

The following proposition is the counterpart of Theorem 14(ii) for equi-join queries.

▶ **Proposition 38.** *For each equi-join query Q and decomposition $\mathcal{T}$ of Q, there are arbitrarily large databases $\mathbf{D}$ such that each cover of $Q(\mathbf{D})$ over $\mathcal{T}$ has size $\Omega(|\mathbf{D}|^{\mathsf{fhtw}(\mathcal{T})})$.*

In Proposition 38, we first construct a natural join query Q' from Q as in Proposition 37. By Theorem 14(ii), there are arbitrarily large databases $\mathbf{D}'$ such that each cover of $Q'(\mathbf{D}')$ over $\mathcal{T}$ has size $\Omega(|\mathbf{D}'|^{\mathsf{fhtw}(\mathcal{T})})$. Given such a database $\mathbf{D}'$, it follows from Proposition 13, that $\Sigma_{B\in\mathcal{S}(\mathcal{T})}|\pi_B Q'(\mathbf{D}')| = \Omega(|\mathbf{D}'|^{\mathsf{fhtw}(\mathcal{T})})$, hence, $\max_{B\in\mathcal{S}(\mathcal{T})}\{|\pi_B Q'(\mathbf{D}')|\} = \Omega(|\mathbf{D}'|^{\mathsf{fhtw}(\mathcal{T})})$. We convert the database $\mathbf{D}'$ into a database $\mathbf{D}$ of size $\mathcal{O}(|\mathbf{D}'|)$ such that $|\pi_B Q(\mathbf{D})| \geq |\pi_B Q'(\mathbf{D}')|$ for each $B \in \mathcal{S}(\mathcal{T})$. By Proposition 13 (adapted to equi-join queries), each cover of $Q(\mathbf{D})$ over $\mathcal{T}$ must have size at least $\max_{B\in\mathcal{S}(\mathcal{T})}\{|\pi_B Q(\mathbf{D})|\}$. Since $\max_{B\in\mathcal{S}(\mathcal{T})}\{|\pi_B Q'(\mathbf{D}')|\} = \Omega(|\mathbf{D}'|^{\mathsf{fhtw}(\mathcal{T})})$ and $\max_{B\in\mathcal{S}(\mathcal{T})}\{|\pi_B Q(\mathbf{D})|\} \geq \max_{B\in\mathcal{S}(\mathcal{T})}\{|\pi_B Q'(\mathbf{D}')|\}$, we conclude that each cover of $Q(\mathbf{D})$ over $\mathcal{T}$ is of size $\Omega(|\mathbf{D}'|^{\mathsf{fhtw}(\mathcal{T})}) = \Omega(|\mathbf{D}|^{\mathsf{fhtw}(\mathcal{T})})$.