Generalized Covers for Conjunctive Queries

Paraschos Koutris
University of Wisconsin-Madison, WI, USA

Abstract

Covers of query results were introduced as succinct lossless representations of join query outputs. A cover is a subset of the query result from which we can efficiently enumerate the output with constant delay and linear preprocessing time. However, covers are dependent on a single tree decomposition of the query. In this work, we generalize the notion of a cover to a set of multiple tree decompositions. We show that this generalization can potentially produce asymptotically smaller covers while maintaining the properties of constant-delay enumeration and linear preprocessing time. In particular, given a set of tree decompositions, we can determine exactly the asymptotic size of a minimum cover, which is tied to the notion of entropic width of the query. We also provide a simple greedy algorithm that computes this cover efficiently. Finally, we relate covers to semiring circuits when the semiring is idempotent.

2012 ACM Subject Classification Theory of computation → Database theory

Keywords and phrases Conjunctive Query, tree decomposition, cover

Digital Object Identifier 10.4230/LIPIcs.ICDT.2025.28

1 Introduction

Join queries are one of the fundamental operations in relational databases. However, the output of a join query can be potentially huge when executed over a large database instance. This is particularly relevant when a join output needs to be sent over a network link in a distributed setting, or if the output result needs to be stored before being processed in a downstream task. In such scenarios, it is often desirable to construct concise representations of the query output such that we can reconstruct the query result as efficiently as possible when and where it is needed. Several succinct data structures have been proposed in the literature, including factorized databases [16] and compressed representations [7].

A *cover* is one such succinct and lossless representation of a join output that was introduced by Kara and Olteanu [13]. A cover is simply a subset of the query result that, together with a given tree decomposition of the query, can efficiently reconstruct the full query output. Here, efficient means that after a preprocessing step that is linear to the size of the representation, we can enumerate the output with a constant delay guarantee. One of the key results in [13] is that, given a tree decomposition T of the query and a database instance of size N, we can always produce a cover of size $O(N^{\mathsf{fhw}(T)})$, where $\mathsf{fhw}(T)$ is the *fractional hypertree width* of T. Thus, choosing a tree decomposition of minimum width gives a cover of size $O(N^{\mathsf{fhw}})$ and that can be shown to be asymptotically optimal. This bound is a large improvement over storing the full output, which can be as large as the AGM bound [2], i.e., $\Omega(N^{\rho^*})$ where ρ^* is the fractional edge cover of the query.

The main insight of this work is that it is possible to construct covers of even smaller asymptotic size if we define the cover to depend not on a single decomposition, but on multiple tree decompositions. This is analogous to the PANDA algorithm [14], which improves the runtime of query evaluation from $O(N^{\mathsf{fhw}})$ to $O(N^{\mathsf{subw}})$ by considering simultaneously multiple tree decompositions. Here, subw is the submodular width of the query. Importantly, even though we allow a dependence on multiple tree decompositions, we can still show that the guarantee of linear-time preprocessing with constant delay enumeration remains unchanged. Thus, this generalized notion of a cover produces an even more succinct and efficient representation of the query output.

In fact, we will show in this paper that by considering the (finite) set of all possible tree decompositions, we can construct covers of size $O(N^{\mathsf{entw}})$, where entw is the *entropic width* of the query, a width measure that is at least as small as the submodular width. As a consequence, we show the *existence* of a data structure of size $O(N^{\mathsf{entw}})$ from which we can enumerate the query output with constant delay. We should note here that, unfortunately, the time to construct this data structure can be much larger than $O(N^{\mathsf{entw}})$, and so the cover construction does not produce a faster query evaluation algorithm.

From a theoretical point of a view, this makes progress to the following fundamental question: what is the smallest possible compression of a query result that still guarantees constant-delay enumeration (i.e., fast decompression)? From a practical point of view, the enumeration guarantee is dependent on the number of tree decompositions, which can be exponentially large in the query. However, we show that as we add more decompositions the size of the cover can only improve; hence, even a small set of decompositions can lead to a large improvement in the cover size compared to considering only a single one. The algorithm we present that constructs a small cover – even though it needs access to the full query result – requires only a linear scan over the query result.

Our Contribution. We summarize our contributions as follows:

- We generalize the notion of a cover of a query result to consider multiple tree decompositions (Section 4). We prove several interesting properties for covers, and show that given a cover of size K we can spend $O(|K|)$ preprocessing time to enumerate the query output with constant delay.
- We present a simple greedy algorithm (Section 5) that, given the query output, produces an asymptotically optimal cover in time linear w.r.t. the query output. Moreover, we show an upper bound that uses not only the input size N as a parameter, but any combination of degree constraints, including different cardinalities, functional dependencies, and degree bounds.
- We provide a lower bound (Section 6) that asymptotically matches our upper bound. The lower bound uses a technically interesting connection to disjunctive Datalog rules to find a worst-case instance for a cover.
- Finally, we show in Section 7 how we can use a cover to produce a *semiring circuit* of size linear to the cover size for the provenance polynomial of the query for any idempotent semiring. A semiring circuit can be thought as a factorization of the query output (more precisely, it is a factorization of the polynomial associated with the query output), with the additional flexibility that it can consider different tree decompositions in its internal representation.

2 Related Work

In this section, we present work that relates to covers and in general efficient representations of query results.

Factorized Databases and Circuits. As shown in [13], covers over a single tree decomposition are related to d-representations over the same tree decomposition. d-representations (and f-representations) are also lossless representations of query outputs, which allow for efficient enumeration, aggregation, and even ML model computation [18]. Query outputs can also be concisely stored using circuits [9, 1] – these circuits represent the polynomial corresponding to the join query and can be evaluated under different semiring semantics. As we will show later in the paper, analogous to the connection between covers and d-representations, there is also a direct connection between generalized covers and semiring circuits.

Compressed Representations of Outputs. Recent work has also looked at how we can further decrease the size of a representation by trading off the enumeration delay guarantee [7, 19, 6]. Such a tradeoff allows for asymptotically smaller succinct data structures, but with an increased possible delay when outputting two consecutive outputs. Even though it would be theoretically possible to consider covers with weaker enumeration guarantees, in this work we focus only on constant delay.

Preprocessing Time and Enumeration. Several algorithms on join computation use a two-phase framework where a preprocessing phase is followed by an output enumeration phase [3]. For instance, for any join query with fractional hypertree width fhw, we need $O(N^{\mathsf{fhw}})$ preprocessing time to achieve constant delay enumeration. The intermediate data structure constructed by the preprocessing phase can be also viewed as a succinct representation of the result; however, it is critical in this case to also have an efficient algorithm to construct the data structure. Recent work has looked at preprocessing time/size and enumeration tradeoffs in this setting [11, 12]. In our paper, even though we construct a very small succinct representation, the runtime can exceed the size of the representation and thus does not lead to a better query evaluation algorithm.

3 Preliminaries

In this section, we present useful notation and terminology.

Conjunctive Queries. A *Conjunctive Query* (CQ) Q is an expression associated to a hypergraph $\mathcal{H} = ([n], \mathcal{E})$ where $[n] = \{1, \ldots, n\}$ and a set $U \subseteq [n]$:

$$Q(\mathbf{x}_U) \leftarrow \bigwedge_{e \in \mathcal{E}} R_e(\mathbf{x}_e)$$

where each R_e is a relation of arity $|e|$, the variables $x_1, x_2, \ldots, x_n$ take values in some discrete domain, and $\mathbf{x}_e := (x_i)_{i \in e}$. We say that Q is *Boolean* if $U = \{\}$ and *full* if $U = [n]$.

Given a tuple t over $[n]$ and a subset $S \subseteq [n]$, we will use $t[S]$ to denote the projection of the tuple on the attributes in S.

Tree Decompositions. We recall the notion of a tree decomposition.

▶ **Definition 1** (Tree Decomposition). *A* tree decomposition *of a hypergraph $\mathcal{H}$ is a pair $(\mathcal{T}, \chi)$, where $\mathcal{T}$ is a tree and χ maps each node t of the tree to a subset $\chi(t)$ of $V(\mathcal{H})$, called a bag, such that:*

1. *every hyperedge $e \in E(\mathcal{H})$ is a subset of $\chi(t)$ for some $t \in V(\mathcal{T})$; and*
2. *for every vertex $v \in V(\mathcal{H})$, the set $\{t | v \in \chi(t)\}$ is a non-empty connected subtree of $\mathcal{T}$.*

We say that a tree decomposition is *non-redundant* if no bag is a subset of another; otherwise it is *redundant*. We let $\mathsf{TD}(\mathcal{H})$ be the set of all non-redundant tree decompositions of $\mathcal{H}$. This set is known to be finite; it can be shown that $|\mathsf{TD}(\mathcal{H})| \leqslant n!$, where n is the number of vertices of the hypergraph (Proposition 2.9 [15]).

Entropic Functions. Let n be a positive integer. A function $h : 2^{[n]} \to \mathbb{R}_+$ is called a *set function* on $[n] = \{1, 2, \ldots, n\}$. Given a discrete random variable X with support in $\mathcal{X}$ and probability distribution $p : \mathcal{X} \to [0, 1]$, its *entropy* is defined as $H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$. When the probability distribution is uniform in the support set, then $H(X) = \log |\mathcal{X}|$. Given

a set of random variables $X_1, \ldots, X_k$, the *joint entropy* $H(X_1, \ldots, X_k)$ is defined as the entropy of the random variable Y that represents the tuple $(X_1, \ldots, X_k)$. A set function is an *entropic function* of order n if there exist random variables $A_1, \ldots, A_n$ such that $h(S) = H((A_i)_{i \in S})$ for any $S \subseteq [n]$. We denote by Γ_n^* the set of all entropic functions of order n, and $\overline{\Gamma}_n^*$ the topological closure of Γ_n^*.[1] The important property of $\overline{\Gamma}_n^*$ is that it is a convex cone and hence characterized by the (infinitely many) linear inequalities it satisfies.

Entropic Width. Let DC be a set of triples $(X, Y, N_{Y|X})$ for some $X \subset Y \subseteq [n]$ and $N_{Y|X} \in \mathbb{N}$ that encodes a set of *degree constraints*. A database instance I over the relational schema $\{R_e\}_{e \in \mathcal{E}}$ satisfies the constraints if for every relation R_e in I with $X \subseteq Y \subseteq e$ and every tuple t defined over schema X, we have $|\pi_Y(R_e \ltimes t)| \leqslant N_{Y|X}$. A constraint of the form $(\varnothing, e, N_e)$ is simply a cardinality constraint that says that relation R_e has size at most N_e. The degree constraints on an instance can be translated as constraints on entropic functions as follows:

$$\mathsf{HDC} := \left\{ h : 2^{[n]} \to \mathbb{R}_+ \mid \bigwedge_{(X,Y,N_{Y|X}) \in \mathsf{DC}} h(Y|X) \leqslant \log N_{Y|X} \right\}$$

where $h(Y|X) := h(Y) - h(X)$. For a given set of degree constraints HDC, we define the "scaled-up" degree constraints $\mathsf{HDC} \times k$ as the same set of constraints but with all the degree bounds multiplied by k. It will be also helpful to define the entropic constraints in the case we are only given a uniform cardinality constraint (input size):

$$\mathsf{ED} := \{h : 2^{[n]} \to \mathbb{R}_+ \mid h(e) \leqslant 1, \forall e \in \mathcal{E}\}$$

We now define the *entropic width* (defined in [9]) and *degree-aware entropic width* (defined in [15] as eda-subw) w.r.t. a set of tree decompositions $\mathbb{T}$ respectively:

$$\mathsf{entw}_{\mathbb{T}}(\mathcal{H}, \mathsf{ED}) := \max_{h \in \overline{\Gamma}_n^* \cap \mathsf{ED}} \min_{(\mathcal{T},\chi) \in \mathbb{T}} \max_{v \in V(\mathcal{T})} h(\chi(v))$$

$$\mathsf{da\text{-}entw}_{\mathbb{T}}(\mathcal{H}, \mathsf{HDC}) := \max_{h \in \overline{\Gamma}_n^* \cap \mathsf{HDC}} \min_{(\mathcal{T},\chi) \in \mathbb{T}} \max_{v \in V(\mathcal{T})} h(\chi(v))$$

When $\mathbb{T} = \mathsf{TD}(\mathcal{H})$, we obtain the standard notions of (degree-aware) entropic width.

Computational Model. We use the uniform-cost RAM model where data values as well as pointers to databases are of constant size. Our runtime analysis will consider data complexity (where the query is considered fixed) unless otherwise stated.

4 Generalized Covers

We start by generalizing the notion of a cover of a query result from one to multiple tree decompositions.

▶ **Definition 2** (Cover). *Let Q be a full CQ with hypergraph $\mathcal{H}$, D be an instance, and $\mathbb{T}$ be a finite set of tree decompositions of $\mathcal{H}$. A relation K over schema $[n]$ is a* cover *of $Q(D)$ w.r.t. $\mathbb{T}$ if*

$$\bigcup_{(\mathcal{T},\chi) \in \mathbb{T}} \bowtie_{t \in V(\mathcal{T})} (\pi_{\chi(t)} K) = Q(D).$$

[1] The topological closure of a set S is defined as the smallest closed set that contains S.

In other words, if for every decomposition we project the cover to its bags and then join the bags together, the union of these outputs must form exactly $Q(D)$. When the set $\mathbb{T}$ consists of a single tree decomposition, then we recover the notion of a cover as in [13]. In our more general definition, the reconstruction of the output $Q(D)$ is based not on a single tree decomposition, but a set of tree decompositions.

▶ **Proposition 3.** *Let Q be a full CQ, D be an instance, and $\mathbb{T}$ be a set of tree decompositions of Q. If K is a cover of $Q(D)$ w.r.t. $\mathbb{T}$, then $K \subseteq Q(D)$.*

Proof. This follows immediately from the fact that for every tree decomposition $(\mathcal{T}, \chi)$ and relational instance K, we have $K \subseteq \bowtie_{t \in V(\mathcal{T})} (\pi_{\chi(t)} K)$. ◀

▶ Remark 4. It is useful to contrast with the original definition of a cover used in [13] for a single tree decomposition $(\mathcal{T}, \chi)$. In this definition, a cover K was equivalently defined as a set of tuples for which $\pi_{\chi(v)}(K) = \pi_{\chi(v)}(Q(D))$ for every $v \in V(\mathcal{T})$. However, such an equivalent characterization does not hold anymore in the case of multiple tree decompositions, since each decomposition may be responsible to produce a (strict) subset of the query output.

We say that a cover K is *minimal* w.r.t. $\mathbb{T}$ if there is no strict subset $K' \subsetneq K$ that is also a cover w.r.t. $\mathbb{T}$ [2]. A cover is *minimum* w.r.t. $\mathbb{T}$ if it has the smallest possible size across all covers w.r.t. $\mathbb{T}$.

▶ **Example 5.** We will use as an example the 4-cycle query:

$$Q(x_1, x_2, x_3, x_4) \leftarrow R_1(x_1, x_2), R_2(x_2, x_3), R_3(x_3, x_4), R_4(x_4, x_1).$$

This CQ has only two non-redundant tree decompositions: $T_1 = \{1,2,3\}, \{1,3,4\}$ and $T_2 = \{1,2,4\}, \{2,3,4\}$. We consider the following relational instance D, where each relation has $2n$ tuples and thus the input size is $|D| = \Theta(n)$. Note also that $|Q(D)| = 2n^2$, hence the output size is quadratic w.r.t. the input size.

R_1	
x_1	x_2
a_1^1	a_2
a_1^2	a_2
$\vdots$	$\vdots$
a_1^n	a_2
b_1	b_2^1
b_1	b_2^2
$\vdots$	$\vdots$
b_1	b_2^n

R_2	
x_2	x_3
a_2	a_3^1
a_2	a_3^2
$\vdots$	$\vdots$
a_2	a_3^n
b_2^1	b_3
b_2^2	b_3
$\vdots$	$\vdots$
b_2^n	b_3

R_3	
x_3	x_4
a_3^1	a_4
a_3^2	a_4
$\vdots$	$\vdots$
a_3^n	a_4
b_3	b_4^1
b_3	b_4^2
$\vdots$	$\vdots$
b_3	b_4^n

R_4	
x_4	x_1
a_4	a_1^1
a_4	a_1^2
$\vdots$	$\vdots$
a_4	a_1^n
b_4^1	b_1
b_4^2	b_1
$\vdots$	$\vdots$
b_4^n	b_1

Suppose now that we want to find a cover w.r.t. $\{T_1\}$ only. Using a result from [13], we know that the size of any cover w.r.t. a single decomposition is at least the size of the largest bag in the decomposition. For T_1, both bags $\{1,2,3\}$ and $\{1,3,4\}$ have size $\Theta(n^2)$, hence the cover has size $\Omega(n^2)$. A similar reasoning provides a lower bound of $\Omega(n^2)$ for the size of any cover w.r.t. T_2. Hence, no matter which tree decomposition we use, a cover w.r.t. a single tree decomposition has size $\Omega(n^2)$.

[2] We should note here that in [13] a cover always refers to a minimal cover. In this paper, however, we will also consider covers that may not be minimal.

Now, suppose we want to find a cover w.r.t. $\{T_1, T_2\}$, i.e., we will include both non-redundant tree decompositions. We claim that the following set K of size only $O(n)$ is a cover w.r.t. $\{T_1, T_2\}$:

K			
x_1	x_2	x_3	x_4
a_1^1	a_2	a_3^1	a_4
$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_1^n	a_2	a_3^n	a_4
b_1	b_2^1	b_3	b_4^1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
b_1	b_2^n	b_3	b_4^n

Indeed, partition $D = D^a \cup D^b$, where D^a, D^b contain the tuples with a, b -values respectively, and similarly partition K into K^a, K^b. Then, one can verify that $\pi_{124}(K^a) \bowtie \pi_{234}(K^a) = Q(D^a)$ and $\pi_{123}(K^b) \bowtie \pi_{134}(K^b) = Q(D^b)$. In other words, we use decomposition T_2 to guide the covering of the a-tuples, and T_1 to guide the covering of the b-tuples. Note that $\pi_{123}(K^a) \bowtie \pi_{134}(K^a) \subsetneq Q(D^a)$.

As we will show later in the paper, for the 4-cycle query we can always produce a cover of size $O(|D|^{3/2})$ using both tree decompositions. In this example, we are able to do even better and produce a cover of only linear size.

4.1 Basic Properties of Covers

The following propositions establish some basic facts about covers.

▶ **Proposition 6.** *Let Q be a full CQ, D be an instance, and $\mathbb{T}, \mathbb{T}'$ be two sets of tree decompositions of Q such that $\mathbb{T}' \supseteq \mathbb{T}$. If K is a cover of $Q(D)$ w.r.t. $\mathbb{T}$, then K is a cover of $Q(D)$ w.r.t. $\mathbb{T}'$ as well.*

Proof. Indeed, we can write the following:

$$Q(D) = \bigcup_{(\mathcal{T},\chi)\in\mathbb{T}} \bowtie_{v\in V(\mathcal{T})} \pi_{\chi(v)}K \subseteq \bigcup_{(\mathcal{T},\chi)\in\mathbb{T}'} \bowtie_{v\in V(\mathcal{T})} \pi_{\chi(v)}K$$

$$\bigcup_{(\mathcal{T},\chi)\in\mathbb{T}'} \bowtie_{v\in V(\mathcal{T})} \pi_{\chi(v)}K \subseteq \bigcup_{(\mathcal{T},\chi)\in\mathbb{T}'} \bowtie_{v\in V(\mathcal{T})} (\pi_{\chi(v)}Q(D)) = Q(D)$$

In the last equation, the first inequality follows from Lemma 3 that implies $K \subseteq Q(D)$, while the last equality follows from the fact that for every tree decomposition $(\mathcal{T}, \chi)$, $\bowtie_{v\in V(\mathcal{T})} (\pi_{\chi(v)}Q(D)) = Q(D)$. ◀

The above proposition tells us that if we add more tree decompositions in a set $\mathbb{T}$, the minimum cover can never increase in size. But how large does $\mathbb{T}$ need to be to achieve the smallest possible cover (among all possible tree decomposition sets)?

▶ **Proposition 7.** *Let Q be a full CQ, D be an instance, and $\mathbb{T}$ be a finite set of tree decompositions of Q. If K covers $Q(D)$ w.r.t. $\mathbb{T}$, then it covers $Q(D)$ w.r.t. the (finite) set of all non-redundant tree decompositions $\mathsf{TD}(\mathcal{H})$.*

Proof. From Proposition 6, we have that K covers $Q(D)$ w.r.t. $\mathbb{T} \cup \mathsf{TD}(\mathcal{H})$. Hence:

$$\bigcup_{(\mathcal{T},\chi)\in\mathbb{T}\cup\mathsf{TD}(\mathcal{H})} \bowtie_{v\in V(\mathcal{T})} \pi_{\chi(v)}K = Q(D)$$

To complete the proof, we will show that

$$\bigcup_{(\mathcal{T},\chi)\in\mathsf{TD}(\mathcal{H})} \bowtie_{v\in V(\mathcal{T})} \pi_{\chi(v)}K = \bigcup_{(\mathcal{T},\chi)\in\mathbb{T}\cup\mathsf{TD}(\mathcal{H})} \bowtie_{v\in V(\mathcal{T})} \pi_{\chi(v)}K$$

It is straightforward that the left-hand-side is contained in the right-hand-side. For the other direction, let $(\mathcal{T},\chi)$ be a redundant tree decomposition in $\mathbb{T}$. Then, there exists a non-redundant tree decomposition $(\mathcal{T}',\chi')$ in $\mathsf{TD}(\mathcal{H})$ such that for every bag $v' \in V(\mathcal{T}')$, there exists a bag $v \in V(\mathcal{T})$ such that $\chi(v) = \chi'(v')$ (this is constructed by eliminating the bags that are contained in some other bag). We will show that for any K over schema $[n]$, it holds that $\bowtie_{v\in V(\mathcal{T})} \pi_{\chi(v)}K \subseteq \bowtie_{v\in V(\mathcal{T}')} \pi_{\chi(v)}K$. Indeed, take any tuple $t \in \bowtie_{v\in V(\mathcal{T})} \pi_{\chi(v)}K$. Consider some bag $v' \in V(\mathcal{T}')$. Then, there exists $v \in V(\mathcal{T})$ with $\chi(v') = \chi(v)$. Hence, $\pi_{\chi(v')}(t) = \pi_{\chi(v)}(t) \in \pi_{\chi(v)}(K) = \pi_{\chi(v')}(K)$. This implies that $t \in \bowtie_{v\in V(\mathcal{T}')} \pi_{\chi(v)}K$ as well. ◀

The above proposition says that we can achieve the best possible cover in terms of size if we take our set of tree decompositions to be $\mathsf{TD}(\mathcal{H})$, which is finite.

4.2 From Covers to Constant-Delay Enumeration

In this section, we show an important property of covers. Given a cover K of $Q(D)$ w.r.t. some set of tree decompositions, we show that with preprocessing time only linear w.r.t. $|K|$, it is possible to enumerate the output of a query Q with constant delay. A constant-delay enumeration algorithm means that the time between outputting two consecutive tuples in $Q(D)$ (as well as the time to produce the first tuple, and the time to finish after producing the last tuple) is $O(1)$ w.r.t. data complexity.

▶ **Theorem 8.** *Let Q be a full CQ, D be an instance, and $\mathbb{T}$ be a finite set of tree decompositions of Q. Suppose we are given a cover K for $Q(D)$ w.r.t. $\mathbb{T}$. Then, with preprocessing time $O(|K|)$ we can enumerate $Q(D)$ with constant delay.*

Proof. We first describe the preprocessing phase. We will use K to construct a materialized instance R_B for every bag B in every decomposition of $\mathbb{T}$. In particular, we scan K, and for every tuple $t \in K$ and every decomposition $(\mathcal{T},\chi)$ in $\mathbb{T}$ and every bag $v \in V(\mathcal{T})$, we add the projection $t[\chi(v)]$ to the bag $R_{\chi(v)}$. At this point, since K is a cover for $Q(D)$, we know that $Q(D) = \bigcup_{(\mathcal{T},\chi)\in\mathbb{T}} \bowtie_{v\in V(\mathcal{T})} R_{\chi(v)}$.

Once we have materialized each bag, computing the join $J_{(\mathcal{T},\chi)} = \bowtie_{v\in V(\mathcal{T})} R_{\chi(v)}$ corresponds to computing an acyclic CQ. Indeed, by the definition of a tree decomposition, the query formed by treating each bag as a relation must be acyclic. Thus, we can use the standard result [3] that, with linear preprocessing time $O(\sum_{v\in V(\mathcal{T})} |R_{\chi(v)}|) = O(|K|)$, we can enumerate the results in $J_{(\mathcal{T},\chi)}$ with constant delay. The last thing we need to address is that the same output tuple may be produced via more than one tree decomposition in $\mathbb{T}$. To deal with this, we can apply Cheater's lemma [5] that tells us that we can enumerate the union of a constant number of constant-delay enumeration algorithms with constant delay. Alternatively, we can also apply the method of Durand and Strozecki [8] for enumeration of unions of sets, as done in [4]. ◀

Algorithm 1 Greedy Cover.

Input : output $Q(D)$ of a full CQ Q, set of tree decompositions $\mathbb{T}$
Output : cover of $Q(D)$ w.r.t. $\mathbb{T}$

```
1 K̄ ← ∅
2 foreach t ∈ Q(D) do
3 |   if ∀(T, χ) ∈ 𝕋 : t ∉ ⋈_{v∈V(T)} π_{χ(v)}(K̄) then
4 |   |   K̄ ← K̄ ∪ {t}
5 return K̄
```

The main message of the above theorem is that considering more tree decompositions does not hurt the constant-delay enumeration property of covers if we consider data complexity.

▶ Remark 9. Although the delay is constant w.r.t. data complexity, it depends linearly on the number of tree decompositions $|\mathbb{T}|$; more precisely, the delay will be $O(|\mathbb{T}| \cdot |Q|)$. If we choose to include all non-redunant tree decompositions, i.e., $\mathbb{T} = \mathsf{TD}(\mathcal{H})$, then the delay can become exponentially large in the size of the query, since $\mathsf{TD}(\mathcal{H})$ can be as large as $n!$. However, in practice one may not need to consider all tree decompositions to obtain a concise representation of the output.

5 Finding Small Covers

In this section, we will seek to find the smallest possible cover we can obtain for $Q(D)$ w.r.t. any finite set of tree decompositions $\mathbb{T}$. We will be interested in providing worst-case guarantees on the size, and not an instance-optimal construction of the minimum-sized cover. In particular, we will show the following theorem.

▶ **Theorem 10.** *Given a full CQ Q with hypergraph $\mathcal{H}$ and a database instance D that satisfies the degree constraints* DC*, there exists a cover of $Q(D)$ w.r.t. a finite set of tree decompositions $\mathbb{T}$ of size $O(2^{\textit{da-entw}_{\mathbb{T}}(\mathcal{H},\mathsf{HDC})})$. Moreover, given the output of the query $Q(D)$, we can construct this cover in time $O(|Q(D)|)$.*

When $\mathbb{T} = \mathsf{TD}(\mathcal{H})$, then the size bound simply becomes $O(2^{\mathsf{da\text{-}entw}(\mathcal{H},\mathsf{HDC})})$. To show the above theorem, we will give in the rest of the section a constructive proof, based on a simple greedy algorithm (Algorithm 1). This algorithm is inspired by the construction used in [15] (Proof of Lemma 4.1) to construct small models for disjunctive Datalog rules. Here, we adapt this idea to a CQ and do a direct analysis.

The algorithm starts with an empty set $\bar{K} \leftarrow \varnothing$, and iterates over the tuples $t \in Q(D)$ (in any order). For every tuple $t \in Q(D)$, if there exists a tree decomposition $(\mathcal{T}, \chi) \in \mathbb{T}$ such that $t \in \bowtie_{v \in V(\mathcal{T})} \pi_{\chi(v)}(\bar{K})$ we do nothing; otherwise, we add t to $\bar{K}$. The algorithm terminates when we have visited all output tuples. To bound the running time of Algorithm 1, we can implement the existence check in line 3 in (amortized) constant time per tuple by maintaining a hash table on the projection on each bag $\pi_B(\bar{K})$, and then simply checking that for every $v \in V(\mathcal{T})$, we have $t[\chi(v)] \in \pi_{\chi(v)}(\bar{K})$ (for example, cuckoo hashing [17] gives us constant-time access and amortized constant-time insertions.) Hence, we have the following bound on the running time.

▶ **Lemma 11.** *Algorithm 1 runs in time $O(|Q(D)|)$.*

By construction of $\bar{K}$, we also obtain:

▶ **Lemma 12.** *The output $\bar{K}$ of Algorithm 1 is a cover of $Q(D)$ w.r.t. $\mathbb{T}$.*

Note that if we run Algorithm 1 with two different sets of tree decompositions $\mathbb{T} \subseteq \mathbb{T}'$, the cover produced for $\mathbb{T}'$ will be at least as small as the one for $\mathbb{T}$. However, the delay in the enumeration of the results from the cover will increase. Hence, we essentially have a tradeoff between the compression size (the size of the cover) and time to decompress (the delay).

Let $\mathcal{M}_{\mathbb{T}}$ be the set of all maps $\beta : \mathbb{T} \to 2^{[n]}$ such that $\beta(\mathcal{T}, \chi) = \chi(v)$ for some $v \in V(\mathcal{T})$. Such a map β is called a *bag selector* that chooses a bag from each tree decomposition in $\mathbb{T}$.

▶ **Lemma 13.** *The set $\bar{K}$ can be partitioned as $\{\bar{K}_\beta\}_{\beta \in \mathcal{M}_{\mathbb{T}}}$ such that for any two tuples $t, t' \in \bar{K}_\beta$ it holds that $t[B] \neq t'[B]$ for every $B \in \mathsf{image}(\beta)$.*

Proof. We will construct the sets $\bar{K}_\beta$ following the construction of $\bar{K}$ in Algorithm 1. Initially, all $\bar{K}_\beta$ are empty. Consider the point where a new tuple t is added in $\bar{K}$ in line 4. Then, for every decomposition $(\mathcal{T}, \chi) \in \mathbb{T}$ there must exist some bag $v \in V(\mathcal{T})$ such that $t[\chi(v)] \notin \pi_{\chi(v)}(\bar{K})$ (if there is more than one such bag, we choose one arbitrarily). Consider the bag selector β such that its image is exactly this set of bags, and add t to $\bar{K}_\beta$. It is easy to see that, for every tuple $t' \in \bar{K}_\beta$ and $B \in \mathsf{image}(\beta)$, we have $t'[B] \in \pi_B(\bar{K})$ and thus it must be that $t[B] \neq t'[B]$. ◀

▶ **Example 14.** We show how Algorithm 1 and the above lemma work using Example 5. Recall that we have two tree decompositions: $T_1 = \{1,2,3\}, \{1,3,4\}$ and $T_2 = \{1,2,4\}, \{2,3,4\}$. There are four bag selectors: $\beta_1 = \{123, 124\}$, $\beta_2 = \{123, 234\}$, $\beta_3 = \{134, 124\}$, and $\beta_4 = \{134, 234\}$. To make the exposition simpler, suppose the instance obtains only four output tuples:

$$t_1 = (a_1^1, a_2, a_3^1, a_4), \quad t_2 = (a_1^1, a_2, a_3^2, a_4), \quad t_3 = (a_1^2, a_2, a_3^1, a_4), \quad t_4 = (a_1^2, a_2, a_3^2, a_4)$$

$\bar{K}$ and all $\bar{K}_\beta$ are initially empty. After reading the first tuple, we can assign it to any selector, say $K_{\beta_1} = \{t_1\}$. The second tuple is also added to $\bar{K}$; however, we will not add it to $\bar{K}_{\beta_1}$ because t_1, t_2 agree on the bag 124. Thus, $\bar{K}_{\beta_2} = \{t_2\}$. For t_3, we will add it to $\bar{K}_{\beta_1}$, so $\bar{K}_{\beta_1} = \{t_1, t_3\}$. Finally, t_4 will not be added, since it can be generated via decomposition T_2.

Equipped with the above two lemmas, we can now prove an upper bound on the size of the cover constructed via Algorithm 1.

▶ **Theorem 15.** *If $\bar{K}$ is the output of Algorithm 1, then $|\bar{K}| = O(2^{\mathit{da\text{-}entw}_{\mathbb{T}}(\mathcal{H}, \mathsf{HDC})})$*

Proof. From Lemma 13, we have that $\sum_{\beta \in \mathcal{M}_{\mathbb{T}}} |\bar{K}_\beta| = |\bar{K}|$. Hence, there exists a set $\bar{K}_\beta$ with size $|\bar{K}_\beta| \geqslant |\bar{K}|/|\mathcal{M}_{\mathbb{T}}|$ such that for any two tuples $t, t' \in \bar{K}_\beta$ it holds that $t[B] \neq t'[B]$ for every $B \in \mathsf{image}(\beta)$. The set $\bar{K}_\beta \subseteq Q(D)$ satisfies all degree constraints, since it is a subset of $Q(D)$ which also satisfies all degree constraints. Now, take a probability distribution over $[n]$ that is uniformly random for $\bar{K}_\beta$, that is, each tuple in $\bar{K}_\beta$ is chosen independently with the same probability. Let $\bar{h}$ denote the entropy of this distribution. Note that by construction, $\bar{h} \in \overline{\Gamma}_n^* \cap \mathsf{HDC}$. Moreover, because the projections of the tuples on the bags of $B \in \mathsf{image}(\beta)$ are disjoint, $\log |\bar{K}_\beta| = \bar{h}(B)$.

Now, consider any tree decomposition $(\mathcal{T}, \chi) \in \mathbb{T}$; then, there exists a bag $v \in V(\mathcal{T})$ such that $B = \chi(v) \in \mathsf{image}(\beta)$:

$$\log |\bar{K}_\beta| = \bar{h}(B) \leqslant \max_{v \in V(\mathcal{T})} \bar{h}(\chi(v)).$$

Taking the min over all tree decompositions in $\mathbb{T}$, we can write:

$$\begin{aligned}\log|\bar{K}_\beta| &\leqslant \min_{(\mathcal{T},\chi)\in\mathbb{T}} \max_{v\in V(\mathcal{T})} \bar{h}(\chi(v)) \\ &\leqslant \max_{h\in\overline{\Gamma}^*_n\cap\mathsf{HDC}} \min_{(\mathcal{T},\chi)\in\mathbb{T}} \max_{v\in V(\mathcal{T})} h(\chi(v)) \\ &= \mathsf{da\text{-}entw}_{\mathbb{T}}(\mathcal{H},\mathsf{HDC}).\end{aligned}$$

The bound in the statement follows from the fact that $|\bar{K}_\beta| \geqslant |\bar{K}|/|\mathcal{M}_{\mathbb{T}}|$. ◀

▶ **Corollary 16.** *Given a full CQ Q with hypergraph $\mathcal{H}$ and a database instance D, there exists a cover of $Q(D)$ w.r.t. $\mathsf{TD}(\mathcal{H})$ of size $O(|D|^{\mathsf{entw}(\mathcal{H})})$.*

▶ **Example 17.** Continuing our running example for the 4-cycle query, we know that for this query $\mathsf{entw} = \mathsf{subw} = 3/2$. Hence, Algorithm 1 constructs a cover that has size at most $O(|D|^{3/2})$; for this, it suffices to use as $\mathbb{T}$ the only two non-redundant tree decompositions.

Combining the above corollary with Theorem 8, we obtain that we can construct a data structure of size only $O(|D|^{\mathsf{entw}})$ such that we can provide a constant-delay enumeration guarantee of the output $Q(D)$. Note that the best-known such data structure has size $O(|D|^{\mathsf{subw}})$, where subw is the *submodular width* of the query. However, it holds that $\mathsf{entw} \leqslant \mathsf{subw}$ [14].[3] Of course, we do not know whether we can construct this data structure in time $O(|D|^{\mathsf{entw}})$, since our algorithm needs time linear w.r.t. the output size, which can be asymptotically larger. However, as the next proposition shows, we can construct a (potentially larger) cover in time and size $O(|D|^{\mathsf{subw}})$ using the PANDA algorithm as a blackbox.

▶ **Proposition 18.** *Given a full CQ Q with hypergraph $\mathcal{H}$ and a database instance D, there exists a cover of $Q(D)$ w.r.t. $\mathsf{TD}(\mathcal{H})$ of size $O(|D|^{\mathsf{subw}(\mathcal{H})})$ that can be constructed in time $\tilde{O}(|D|^{\mathsf{subw}(\mathcal{H})})$.*[4]

6 Lower Bounds for Covers

To prove the lower bound, we will relate the size of a cover to the size of a model of a disjunctive Datalog rule, following a similar argument as used in [9] to connect the size of a semiring circuit to disjunctive Datalog. Following the definition in [15], *a disjunctive Datalog rule* over a hypergraph $\mathcal{H} = ([n], \mathcal{E})$ is an expression of the form:

$$P : \bigvee_{B\in\mathcal{B}} T_B(\mathbf{x}_B) \leftarrow \bigwedge_{e\in\mathcal{E}} R_e(\mathbf{x}_e)$$

where $B \subseteq [n]$ and $\mathcal{B} \subseteq 2^{[n]}$. Each output relation T_B on the head of the rule is called a *target*. Given an instance D, a model of a disjunctive Datalog rule is a tuple $\mathbf{T} = (T_B)_{B\in\mathcal{B}}$ of relation instances such that for any tuple t over schema $[n]$, if $t[e] \in R_e$ for all $e \in \mathcal{E}$, then there exists a target T_B such that $t[B] \in T_B$. The size of a model is defined to be $\max_B |T_B|$ and the output size of a disjunctive Datalog rule over an instance D is the minimum size over all models of the rule.

[3] It is not known whether there is some query where there is a gap between the two width measures.
[4] The notation $\tilde{O}$ allows for a polylogarithmic factor in the input size $|D|$.

▶ **Lemma 19.** *Let Q be a full CQ, D be an instance, and $\mathbb{T}$ be a finite set of tree decompositions of Q. Suppose K is a cover of $Q(D)$ w.r.t. $\mathbb{T}$, and let β be any bag selector for $\mathbb{T}$. Consider the disjunctive rule*

$$P_\beta : \bigvee_{B \in \mathsf{image}(\beta)} T_B(\mathbf{x}_B) \leftarrow \bigwedge_{e \in \mathcal{E}} R_e(\mathbf{x}_e).$$

Then, $|K| \geqslant |P_\beta(D)|$.

Proof. We will construct a model of P_β from the cover K by taking $T_B := \pi_B(K)$ for every $B \in \mathsf{image}(\beta)$. Clearly, $|P_\beta| = \max_B |T_B| \leqslant |K|$. To show that $\{T_B\}_{B \in \mathsf{image}(\beta)}$ is a model for P_β, consider any tuple $t \in Q(D)$. Then, since K is a cover, there exists a tree decomposition $(\mathcal{T}, \chi)$ such that $t \in \bowtie_{v \in V(\mathcal{T})} \pi_{\chi(v)}(K)$. Since β is a bag selector, there exists $B \in \mathsf{image}(\beta)$ for which $\beta(\mathcal{T}, \chi) = B$, and for this bag, $t[B] \in \pi_B(K) = T_B$. ◀

▶ **Example 20.** We continue our running example for the 4-cycle query with $\mathbb{T} = \{T_1, T_2\}$. Consider the bag selector β where we choose bag $\{1, 2, 3\}$ from decomposition T_1 and bag $\{1, 2, 4\}$ from decomposition T_2. Then, the following disjunctive Datalog rule is formed:

$$P_\beta : T_{123}(x_1, x_2, x_3) \vee T_{124}(x_1, x_2, x_4) \leftarrow R_1(x_1, x_2), R_2(x_2, x_3), R_3(x_3, x_4), R_4(x_4, x_1).$$

Lemma 19 tells us than any lower bound on the size of the model for P_β is also a lower bound for the size of any cover w.r.t. $\mathbb{T}$. Note that the rule that has only target T_{123} (or T_{124}) does not transfer its lower bound, since the disjunction on the left-hand side needs to include all tree decompositions in $\mathbb{T}$.

▶ **Theorem 21.** *Let Q be a full CQ, and $\mathbb{T}$ be a finite set of tree decompositions of Q. Consider a set of constraints HDC. Then, for any $\epsilon > 0$ there exists a scale factor k and an instance D that satisfies the constraints $\mathsf{HDC} \times k$ for which for every cover K of $Q(D)$ w.r.t. $\mathbb{T}$ we have: $\log |K| \geqslant (1 - \epsilon) \cdot \mathsf{da\text{-}entw}_{\mathbb{T}}(\mathcal{H}, \mathsf{HDC} \times k)$.*

Proof. We will use the lower bound construction for an output of a disjunctive rule [15] and follow the proof of Theorem 3.7 in [9]. Let $\mathbf{B}_{\mathbb{T}}$ be the collection of images of all $\beta \in \mathcal{M}_{\mathbb{T}}$. Then, we can bound $\mathsf{da\text{-}entw}_{\mathbb{T}}(\mathcal{H}, \mathsf{HDC} \times k)$ as follows:

$$\mathsf{da\text{-}entw}_{\mathbb{T}}(\mathcal{H}, \mathsf{HDC} \times k) \leqslant \max_{\mathcal{B} \in \mathbf{B}_{\mathbb{T}}} \max_{h \in \overline{\Gamma}^*_n \cap \mathsf{HDC} \times k} \min_{B \in \mathcal{B}} h(B)$$

For a bag selector β, consider the disjunctive rule P_β as constructed in Lemma 19. We know from Lemma 4.4 in [15] that for any $\epsilon > 0$ there exists an integer $k_\beta > 0$ and an instance I_β such that

$$\log |P(I_\beta)| \geqslant (1 - \epsilon) \max_{h \in \overline{\Gamma}^*_n \cap \mathsf{HDC} \times k_\beta} \min_{B \in \mathcal{B}} h(B).$$

Consider now the k_0 that maximizes k_β over all bag selectors in $\mathcal{M}_{\mathbb{T}}$, and let β_0 the bag selector that maximizes the right-hand side of the first inequality. Then,

$$\log |P(I_{\beta_0})| \geqslant (1 - \epsilon) \cdot \mathsf{da\text{-}entw}_{\mathbb{T}}(\mathcal{H}, \mathsf{HDC} \times k_0).$$

To conclude the proof, observe that from Lemma 19 we have that for a cover K, it holds that $\log |K| \geqslant \log |P(I_{\beta_0})|$. ◀

This lower bound matches (asymptotically) the upper bound in the previous section. When the degree constraints correspond to a uniform cardinality constraint N on each relation and $\mathbb{T}$ is the set of all non-redundant tree decompositions, then we obtain from Theorem 21 that there exists an instance where any cover has size $\Omega(N^{\mathsf{entw}})$.

7 From Covers to Semiring Circuits

In this section, we will show how we can go from a cover of a query result to a semiring circuit. Before we present the main result, we need to introduce some further terminology.

A semiring $\mathbb{S} = (\mathbf{D}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a structure where $\oplus$ is the addition and $\otimes$ is the multiplication such that (i) $(\mathbf{D}, \oplus, \mathbf{0})$ and $(\mathbf{D}, \otimes, \mathbf{1})$ are commutative monoids, (ii) multiplication is distributive over addition, and (iii) $x \otimes \mathbf{0} = \mathbf{0}$ for every element $x \in \mathbf{D}$. The semiring is also *idempotent* if $x \oplus x = x$ for every element $x \in \mathbf{D}$. An example of an idempotent semiring is the tropical semiring $(\mathbb{Z}, \min, +, +\infty, 0)$.

Given a full CQ with hypergraph $\mathcal{H}$, an instance D and a semiring $\mathbb{S}$, we will add to each input tuple t from relation R_e an *annotation* x_t^e: this annotation can be thought as a variable that takes values from the domain $\mathbf{D}$ of the semiring $\mathbb{S}$. For example, if $\mathbb{S}$ is the Boolean semiring $(\{0,1\}, \vee, \wedge, 0, 1)$, the annotation captures the presence or absence of the tuple t. As another example, if $\mathbb{S}$ is the counting semiring $(\mathbb{N}, +, \times, 0, 1)$, the annotation captures the multiplicity of the tuple t. We can now define the sum-product polynomial as:

$$p_D^{\mathcal{H}} := \bigoplus_{t \in Q(D)} \bigotimes_{e \in \mathcal{E}} x^e_{t[e]}.$$

The sum-product polynomial captures different types of provenance of the query, depending on the semiring we use to interpret it [10]. When $\mathbb{S}$ is the Boolean semiring, the polynomial captures Boolean provenance, while if it is the counting semiring, it encodes how-provenance. Why-provenance can also be captured in this framework.

We can concisely represent the sum-product polynomial using circuits. For our purposes, we will define a circuit F over $\mathbb{S}$ as a directed acyclic graph with input nodes the variables $x^e_{t[e]}$ and constants $\mathbf{0}, \mathbf{1}$. Every other node is labelled by a semiring operation ($\oplus$ or $\otimes$) and has fan-in 2. An input gate of F is any gate with fan-in 0 and the output gate of F is the unique gate with fan-out 0.[5] The size of the circuit, denoted as $|F|$, is the number of gates in the circuit. If we have a circuit F that computes the polynomial $p_D^{\mathcal{H}}$, then we can evaluate $p_D^{\mathcal{H}}$ for any input values in time only $O(|F|)$ by evaluating the circuit in a bottom-up fashion.

Our main result in this section shows a tight connection between covers and circuits. In particular, we show that we can use a cover of size K to construct a semiring circuit of the same size $O(K)$. Hence, small covers will lead to small circuits.

▶ **Theorem 22.** *Let Q be a full CQ with hypergraph $\mathcal{H}$, D be an instance, and $\mathbb{T}$ be a finite set of tree decompositions of Q. Let K be a cover of $Q(D)$ w.r.t. $\mathbb{T}$. Then, in time $O(|K|)$ we can construct a semiring circuit of size $O(|K|)$ for the polynomial $p_D^{\mathcal{H}}$ under any idempotent semiring.*

Proof. For a decomposition $T = (\mathcal{T}, \chi) \in \mathbb{T}$, we will first augment it with a mapping $\mu_T : \mathcal{E} \to V(\mathcal{T})$; this mapping assigns each hyperedge of $\mathcal{H}$ to a node v in the tree decomposition. For a node $v \in V(\mathcal{T})$ and a tuple t, we can now define a new variable

$$y_t^v := \bigotimes_{e \in \mathcal{E} : \mu_T(e) = v} x^e_{t[e]}$$

In other words, the new variable corresponds to taking the semiring product of the input tuples for the hyperedges we have assigned to the bag v. If there is no hyperedge assigned to v, then $y_t^v := \mathbf{1}$. Any variable y_t^v can be always encoded via a constant-size subcircuit that computes a constant-size semiring product.

[5] In general, we could define a circuit to have multiple output gates, but for the purposes of this paper it suffices to consider circuits with a unique output gate.

Using the definition of a cover, we can now write:

$$
\begin{aligned}
p_D^{\mathcal{H}} &= \bigoplus_{t \in Q(D)} \bigotimes_{e \in \mathcal{E}} x^e_{t[e]} \\
&= \bigoplus_{t \in \bigcup_{(\mathcal{T},\chi)\in\mathbb{T}} \bowtie_{v\in V(\mathcal{T})}(\pi_{\chi(v)}K)} \left(\bigotimes_{e\in\mathcal{E}} x^e_{t[e]} \right) && \text{(by definition of a cover)} \\
&= \bigoplus_{(\mathcal{T},\chi)\in\mathbb{T}} \left(\bigoplus_{t \in \bowtie_{v\in V(\mathcal{T})}(\pi_{\chi(v)}K)} \left(\bigotimes_{e\in\mathcal{E}} x^e_{t[e]} \right) \right) && \text{(by the idempotence of } \oplus) \\
&= \bigoplus_{(\mathcal{T},\chi)\in\mathbb{T}} \left(\bigoplus_{t \in \bowtie_{v\in V(\mathcal{T})}(\pi_{\chi(v)}K)} \left(\bigotimes_{v\in V(\mathcal{T})} \bigotimes_{e\in\mathcal{E}:\mu(e)=v} x^e_{t[e]} \right) \right) && \text{(by commutativity of } \oplus) \\
&= \bigoplus_{(\mathcal{T},\chi)\in\mathbb{T}} \left(\bigoplus_{t \in \bowtie_{v\in V(\mathcal{T})}(\pi_{\chi(v)}K)} \left(\bigotimes_{v\in V(\mathcal{T})} y^v_t \right) \right) && \text{(by definition of } y^v_t)
\end{aligned}
$$

At this point, we can see that it suffices to build a subcircuit for each tree decomposition $(\mathcal{T}, \chi) \in \mathbb{T}$ and then sum their output gates to obtain the final output. Since $|\mathbb{T}|$ is data-independent, the number of these sum-gates is a constant. Hence, our problem reduces to constructing a circuit for the polynomial

$$p_T := \bigoplus_{t \in \bowtie_{v\in V(\mathcal{T})}(\pi_{\chi(v)}K)} \left(\bigotimes_{v\in V(\mathcal{T})} y^v_t \right).$$

This corresponds to constructing a circuit for the polynomial of an acyclic query with body $\bigwedge_{v\in V(\mathcal{T})} S_{\chi(v)}(\mathbf{x}_{\chi(v)})$, where each input relation $S_{\chi(v)}$ is constructed by computing the projection on the cover $\pi_{\chi(v)}K$. Any such projection can be computed in linear time $O(|K|)$. Additionally, from [9] (Theorem 4.1), we know how to construct this circuit in time linear w.r.t. the size of the input, which is $O(|K|)$. Moreover, the size of the circuit is bounded by the input size, and thus is also $O(|K|)$. ◀

▶ Remark 23. The idempotence of the semiring is a critical requirement for the above theorem, since otherwise the proof breaks. Indeed, if the same output tuple is produced by multiple tree decompositions, then it is not possible to correctly split to decompositions with $\oplus$. Thus, we cannot use a cover to construct a semiring circuit for, say, the counting semiring $(\mathbb{N}, +, \times, 0, 1)$ which is not idempotent.

8 Conclusion

In this paper, we have shown that by generalizing covers to depend on multiple tree decompositions, we can construct covers of asymptotically smaller size. We have also provided matching worst-case lower bounds for general degree constraints.

Several research questions on covers remain open. A first question is how efficiently we can construct a small cover of size $O(|D|^{\mathsf{entw}})$. Our current construction takes time $O(|Q(D)|)$, which can be as large as $\Omega(|D|^{\rho^*})$. It would be interesting to explore whether it is possible to reduce this time down to $O(|D|^{\mathsf{subw}})$ using the PANDA algorithm as a blackbox.

A second direction is to consider algorithms that compute the instance-optimal cover (instead of the worst-case optimal). It is likely that this becomes a computationally hard problem – especially when we consider multiple tree decompositions – however, there might be cases where the problem of finding the smallest cover is tractable.

Finally, it is possible to consider a variant of a generalized cover where the output of each decomposition is disjoint from each other. In other words, we want $Q(D)$ to be partitioned into $\{I_i\}_{i\in\mathbb{T}}$ such that each I_i has a cover K_i w.r.t. the decomposition i. Such a "disjoint cover" $\{K_i\}_{i\in\mathbb{T}}$ would allow us to construct a semiring circuit for any semiring, even one that is non-idempotent. The current greedy algorithm, however, does not have the disjointness guarantee and thus it is not clear how we can construct disjoint covers of small size.

References

1 Antoine Amarilli and Florent Capelli. Tractable circuits in database theory. *SIGMOD Rec.*, 53(2):6–20, 2024. doi:10.1145/3685980.3685982.

2 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. In *FOCS*, pages 739–748. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.43.

3 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2007. doi:10.1007/978-3-540-74915-8_18.

4 Christoph Berkholz and Nicole Schweikardt. Constant delay enumeration with fpt-preprocessing for conjunctive queries of bounded submodular width. In *MFCS*, volume 138 of *LIPIcs*, pages 58:1–58:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.MFCS.2019.58.

5 Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. *ACM Trans. Database Syst.*, 46(2):5:1–5:41, 2021. doi:10.1145/3450263.

6 Shaleen Deep, Xiao Hu, and Paraschos Koutris. General space-time tradeoffs via relational queries. In *WADS*, volume 14079 of *Lecture Notes in Computer Science*, pages 309–325. Springer, 2023. doi:10.1007/978-3-031-38906-1_21.

7 Shaleen Deep and Paraschos Koutris. Compressed representations of conjunctive query results. In *PODS*, pages 307–322. ACM, 2018. doi:10.1145/3196959.3196979.

8 Arnaud Durand and Yann Strozecki. Enumeration complexity of logical query problems with second-order variables. In *CSL*, volume 12 of *LIPIcs*, pages 189–202. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPICS.CSL.2011.189.

9 Austen Z. Fan, Paraschos Koutris, and Hangdong Zhao. Tight bounds of circuits for sum-product queries. *Proc. ACM Manag. Data*, 2(2):87, 2024. doi:10.1145/3651588.

10 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40. ACM, 2007. doi:10.1145/1265530.1265535.

11 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in static and dynamic evaluation of hierarchical queries. In *PODS*, pages 375–392. ACM, 2020. doi:10.1145/3375395.3387646.

12 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Evaluation trade-offs for acyclic conjunctive queries. In *CSL*, volume 252 of *LIPIcs*, pages 29:1–29:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CSL.2023.29.

13 Ahmet Kara and Dan Olteanu. Covers of query results. In *ICDT*, volume 98 of *LIPIcs*, pages 16:1–16:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICDT.2018.16.

14 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do Shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *PODS*, pages 429–444. ACM, 2017. doi:10.1145/3034786.3056105.

15 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another?, 2023. arXiv:1612.02503.

16 Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Trans. Database Syst.*, 40(1):2:1–2:44, 2015. doi:10.1145/2656335.

17 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *ESA*, volume 2161 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2001. doi:10.1007/3-540-44676-1_10.

18 Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *SIGMOD Conference*, pages 3–18. ACM, 2016. doi:10.1145/2882903.2882939.

19 Hangdong Zhao, Shaleen Deep, and Paraschos Koutris. Space-time tradeoffs for conjunctive queries with access patterns. In *PODS*, pages 59–68. ACM, 2023. doi:10.1145/3584372.3588675.

A Appendix

We show here the proof of Proposition 18.

Proof. Consider $\mathbf{B}_{\mathsf{TD}}$ and for each $\mathcal{B} \in \mathbf{B}_{\mathsf{TD}}$, the disjunctive Datalog rule

$$P : \bigvee_{B \in \mathcal{B}} T_B(\mathbf{x}_B) \leftarrow \bigwedge_{e \in \mathcal{E}} R_e(\mathbf{x}_e)$$

From the proof of Proposition 7.13 in [15], we can construct in time $\tilde{O}(|D|^{\mathsf{subw}(\mathcal{H})})$ a model $(T_B)_{B \in \mathcal{B}}$ of size $O(|D|^{\mathsf{subw}(\mathcal{H})})$ using the PANDA algorithm.

Next, at each bag $\chi(v)$ in every tree decomposition $(\mathcal{T}, \chi) \in \mathsf{TD}$, we construct a table $S_{\chi(v)}(\mathbf{x}_{\chi(v)})$ ($\mathbf{x}_{\chi(v)}$ is its schema) as follows: (1) take the union over all output tables $T_{\chi(v)}(\mathbf{x}_{\chi(v)})$ from every disjunctive Datalog rule defined by $\mathcal{B} = \mathsf{image}(\beta) \in \mathbf{B}_{\mathsf{TD}}$, if the bag selector β selects this bag $\chi(v)$ from $(\mathcal{T}, \chi)$, and (2) semijoin-reduce the union by every input relation $R_e(\mathbf{x}_e)$, to prune off tuples that do not contribute to the output. It is easy to verify that $S_{\chi(v)}(\mathbf{x}_{\chi(v)})$ is of size $O(|D|^{\mathsf{subw}(\mathcal{H})})$.

By Corollary 7.13 in [15], the query results can be written as the following union of CQs, where we have one CQ per tree decomposition:

$$Q(D) = \bigcup_{(\mathcal{T},\chi) \in \mathsf{TD}} \bowtie_{v \in V(\mathcal{T})} S_{\chi(v)}(\mathbf{x}_{\chi(v)})$$

Let $Q_{(\mathcal{T},\chi)} := \bowtie_{v \in V(\mathcal{T})} S_{\chi(v)}(\mathbf{x}_{\chi(v)})$. Note that this corresponds to computing an acyclic join over relations with sizes bounded by $O(|D|^{\mathsf{subw}(\mathcal{H})})$. We can now use the standard construction of a cover (Lemma 23 in [13]) to construct in time $\tilde{O}(|D|^{\mathsf{subw}(\mathcal{H})})$ a cover $K_{(\mathcal{T},\chi)}$ of $Q_{(\mathcal{T},\chi)}$ of size $O(|D|^{\mathsf{subw}(\mathcal{H})})$. Finally, to obtain the cover K we simply take the union of all covers, i.e., $K := \bigcup_{(\mathcal{T},\chi)} K_{(\mathcal{T},\chi)}$. ◀