



HAL
open science

Linear Programs with Conjunctive Database Queries

Florent Capelli, Nicolas Crosetti, Joachim Niehren, Jan Ramon

► **To cite this version:**

Florent Capelli, Nicolas Crosetti, Joachim Niehren, Jan Ramon. Linear Programs with Conjunctive Database Queries. Logical Methods in Computer Science, inPress, Volume 20, Issue 1, 10.46298/lmcs-20(1:9)2024 . hal-04317553

HAL Id: hal-04317553

<https://hal.science/hal-04317553>

Submitted on 12 Dec 2023




HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

LINEAR PROGRAMS WITH CONJUNCTIVE DATABASE QUERIES

FLORENT CAPELLI ^{a,b}, NICOLAS CROSETTI ^b, JOACHIM NIEHREN ^b, AND JAN RAMON ^b

^a Université de Lille
e-mail address: florent.capelli@univ-lille.fr

^b Inria, Lille
e-mail address: nicolas.crosetti@inria.fr, joachim.niehren@inria.fr, jan.ramon@inria.fr

ABSTRACT. In this paper, we study the problem of optimizing a linear program whose variables are the answers to a conjunctive query. For this we propose the language LP(CQ) for specifying linear programs whose constraints and objective functions depend on the answer sets of conjunctive queries. We contribute an efficient algorithm for solving programs in a fragment of LP(CQ). The natural approach constructs a linear program having as many variables as there are elements in the answer set of the queries. Our approach constructs a linear program having the same optimal value but fewer variables. This is done by exploiting the structure of the conjunctive queries using generalized hypertree decompositions of small width to factorize elements of the answer set together. We illustrate the various applications of LP(CQ) programs on three examples: optimizing deliveries of resources, minimizing noise for differential privacy, and computing the s-measure of patterns in graphs as needed for data mining.

CONTENTS

| | |
|--------------------------------|---|
| 1. Introduction | 2 |
| 1.1. A Concrete Example | 3 |
| 1.2. Related Work | 5 |
| 1.3. Organization of the paper | 6 |
| 2. Preliminaries | 6 |
| 2.1. Linear Programs | 7 |
| 2.2. Conjunctive Queries | 8 |
| 2.3. Relational Databases | 9 |

Key words and phrases: Database, Linear program, optimization.

This work was partially supported by the French Agence Nationale de la Recherche, AGGREG project reference ANR-14-CE25-0017-01, Headwork project reference ANR-16-CE23-0015, KCODA project ANR-20-CE48-0004 and by a grant of the Conseil Régional Hauts-de-France. The project DATA, Ministère de l'Enseignement Supérieur et de la Recherche, Région Nord-Pas de Calais and European Regional Development Fund (FEDER) are acknowledged for supporting and funding this work. We also thank Sylvain Salvati, Sophie Tison and Yuyi Wang for fruitful discussions and anonymous reviewers of a previous version of this paper for their helpful comments.

| | |
|--|----|
| 2.4. Hypertree Decompositions | 9 |
| 3. Linear Programs with Closed Weight Expressions | 11 |
| 3.1. Example | 12 |
| 3.2. Complexity of solving $LP_{clos}(CQ_{\Sigma})$ | 13 |
| 3.3. Replacement Rewriting | 14 |
| 3.4. Interpretations of linear programs | 16 |
| 4. Solving $LP_{clos}(CQ_{\Sigma})$ linear programs efficiently | 18 |
| 4.1. Tree decomposition of $LP_{clos}(CQ_{\Sigma})$ | 19 |
| 4.2. Factorized Interpretation of quantifier free $LP_{clos}(CQ_{\Sigma})$ | 19 |
| 4.3. Linear Programs with Existentially Quantified Conjunctive Queries | 23 |
| 5. Linear Programs with Open Weight Expressions | 26 |
| 5.1. Closure and semantics | 28 |
| 5.2. Complexity of solving $LP(CQ_{\Sigma})$ programs | 30 |
| 5.3. Case study | 33 |
| 6. Weightings on Tree Decompositions | 34 |
| 6.1. Factorized interpretation and weightings | 35 |
| 6.2. Constructing Weightings | 36 |
| 7. Applications | 42 |
| 7.1. Minimizing Noise for ε -Differential Privacy. | 42 |
| 7.2. Computing the s-Measure for Graph Pattern Matching. | 44 |
| 8. Conclusion and future work | 46 |
| References | 46 |
| Appendix A. Proofs of Section 6.2 | 48 |
| A.1. Proof of Lemma 6.2 | 48 |
| A.2. Proof of Lemma 6.3 | 48 |
| A.3. Proof of Lemma 6.4 | 48 |

1. INTRODUCTION

When modeling optimization problems it often seems natural to separate the logical constraints from the relational data. This holds for linear programming with AMPL [FGK90] and for constraint programming in MiniZinc [NSB⁺07]. It was also noticed in the context of database research, when using integer linear programming for finding optimal database repairs as proposed by Kolaitis, Pema and Tan [KPT13], or when using linear optimization to explain the result of a database query to the user as proposed by Meliou and Suciu [MS12]. Moreover, tools like `SolveDB` [ŠP16] have been developed to better integrate mixed integer programming and thus linear programming into relational databases.

We also find it natural to define the relational data of linear optimization problems by database queries. For this reason, we propose the language of linear programs with conjunctive queries $LP(CQ_{\Sigma})$ in the present paper. An $LP(CQ_{\Sigma})$ program is a linear program with constructs allowing to express linear constraints and linear sums over the weightings of an answer set of database queries. It hence allows us to express an optimization problem with a linear objective function subject to linear constraints that are parameterized by conjunctive queries. To do so, we define the *natural interpretation* $\langle L \rangle^{\mathbb{D}}$ of an $LP(CQ_{\Sigma})$

L over a database \mathbb{D} that is a linear program whose variables are in correspondence with the answer set of the queries of L , hence, $\langle L \rangle^{\mathbb{D}}$ expresses an optimization problem whose solutions are weightings of the answer sets of conjunctive queries. The optimal weightings of $LP(CQ_{\Sigma})$ programs can be computed in a natural manner, by first answering the database queries, then generating the interpretation of L over \mathbb{D} and solving it by calling a linear solver. We then approach the question – to our knowledge for the first time – of whether this can be done with lower complexity for subclasses of conjunctive queries such as the class of acyclic conjunctive queries.

As our main contribution we present a more efficient algorithm for computing the optimal value of a program in $LP(CQ_{\Sigma})$ programs that is able to exploit hypertree decomposition of the queries to speed up the computation. Our algorithm operates in two phases: first, it unfolds universal quantifiers present in $LP(CQ_{\Sigma})$ programs to generate a program in a more restrictive language that we call $LP_{clos}(CQ_{\Sigma})$. Then, the algorithm exploits a hypertree decomposition to construct an alternate interpretation of an $LP_{clos}(CQ_{\Sigma})$ program over a database that we call the factorized interpretation. The factorized interpretation is a linear program having the same optimal value as the linear program resulting in the natural interpretation of $LP(CQ_{\Sigma})$ while being more succinct. It uses different linear program variables that intuitively represent sums of the linear program variables in the natural interpretation. The number of linear program variables in the factorized interpretation depends only on the fractional hypertree width of hypertree decompositions of the queries provided in the input, rather than on the number of query variables. In this manner, our more efficient algorithm can decrease the data complexity, i.e., the degree of the polynomial in the upper bound of the run time of the naive algorithm based on computing the natural interpretation and solving it with a linear program solver. With respect to the combined complexity, even solving $LP_{clos}(CQ_{\Sigma})$ programs is NP-hard and coNP-hard in general, but our approach shows that some cases are tractable.

We prove the correctness of the factorized interpretation with respect to the natural interpretation – that is, the fact that factorized and natural interpretation generates linear program with the same optimal value – by exhibiting a correspondence between weightings of answer sets on the natural interpretation, and weightings of answer sets on the factorized interpretation. This correspondence can be seen as an independent contribution as it shows that one can reconstruct a relevant weighting of the answer set of a quantifier free conjunctive query by only knowing the value of the projected weighting on the bags of a tree decomposition. Conjunctive queries with existential quantifiers are dealt with by showing that one can find an equivalent projecting $LP(CQ_{\Sigma})$ program with quantifier free conjunctive queries only.

1.1. A Concrete Example. We start by illustrating the language $LP(CQ_{\Sigma})$ on an example.

Resource Delivery Optimization. We consider a situation in logistics where a company received orders for specific quantities of resource objects. The objects must be produced at a factory, then transported to a warehouse before being delivered to the buyer. The objective is to fulfill every order while minimizing the overall delivery costs and respecting the production capacities of the factories as well as the storing capacities of the warehouses.

Let F be the set of factories, O the set of objects, W the set of warehouses and B the set of buyers. We consider a database \mathbb{D} with elements in the domain $D = F \uplus O \uplus W \uplus B \uplus \mathbb{R}^+$.

The database \mathbb{D} has four tables. The first table $prod^{\mathbb{D}} \subseteq \mathbf{F} \times \mathbf{O} \times \mathbb{R}^+$ contains triples (f, o, q) stating that the factory f can produce up to q units of object o . The second table $order^{\mathbb{D}} : \mathbf{B} \times \mathbf{O} \times \mathbb{R}^+$ contains triples (b, o, q) stating that the buyer b orders q units of object o . The third table $store^{\mathbb{D}} \subseteq \mathbf{W} \times \mathbb{R}^+$ contains pairs (w, l) stating that the warehouse w has a storing limit of l . The fourth table $route^{\mathbb{D}} : (\mathbf{F} \times \mathbf{W} \times \mathbb{R}^+) \cup (\mathbf{W} \times \mathbf{B} \times \mathbb{R}^+)$ contains triples (f, w, c) stating that the transport from factory f to warehouse w costs c , and triples (w, b, c) stating that the transport from warehouse w to buyer b costs c . The query:

$$dlr(f, w, b, o) = \exists q. \exists q_2. \exists c. \exists c_2. prod(f, o, q) \wedge order(b, o, q_2) \wedge route(f, w, c) \wedge route(w, b, c_2)$$

selects from the database \mathbb{D} all tuples (f, w, b, o) such that the factory f can produce some objects o to be delivered to buyer b through the warehouse w . Let $Q = dlr(f', w', b', o')$ and let $sol^{\mathbb{D}}(Q)$ be the answers of Q on database \mathbb{D} . The goal is to determine for each of these possible deliveries the quantity of the object that should actually be sent. These quantities are modeled by the unknown weights θ_Q^α of the query answers $\alpha \in sol^{\mathbb{D}}(Q)$. For any factory f and warehouse w the sum $\sum_{\alpha \in sol^{\mathbb{D}}(Q \wedge w' \doteq w \wedge f' \doteq f)} \theta_Q^\alpha$ is described by the expression **weight** $_{f' \doteq f \wedge w' \doteq w}(Q)$ when interpreted over \mathbb{D} .

We use the $LP(CQ_{\Sigma})$ program in Figure 1 to describe the optimal weights that minimize the overall delivery costs.

$$\begin{aligned} \text{minimize} \quad & \sum_{(f,w,c):route(f,w,c)} \mathbf{num}(c) \mathbf{weight}_{\mathbf{x}:f' \doteq f \wedge w' \doteq w}(Q) + \\ & \sum_{(w,b,c):route(w,b,c)} \mathbf{num}(c) \mathbf{weight}_{\mathbf{x}:w' \doteq w \wedge b' \doteq b}(Q) \\ \text{subject to} \quad & \forall (f, o, q):prod(f, o, q). \mathbf{weight}_{\mathbf{x}:f' \doteq f \wedge o' \doteq o}(Q) \leq \mathbf{num}(q) \wedge \\ & \forall (b, o, q):order(b, o, q). \mathbf{weight}_{\mathbf{x}:b' \doteq b \wedge o' \doteq o}(Q) \geq \mathbf{num}(q) \wedge \\ & \forall (w, l):store(w, l). \mathbf{weight}_{\mathbf{x}:w' \doteq w}(Q) \leq \mathbf{num}(l) \end{aligned}$$

Figure 1: A $LP(CQ_{\Sigma})$ program for the resource delivery optimization where $Q = dlr(\mathbf{x})$ and $\mathbf{x} = (f', w', b', o')$.

The weights depend on the interpretation of the program over the database, since \mathbb{D} specifies the production capacities of the factories, the stocking limits of the warehouses, etc. The program has the following constraints:

- for each $(f, o, q) \in prod^{\mathbb{D}}$ the overall quantity of object o produced by f is at most q ¹.
- for each $(b, o, q) \in order^{\mathbb{D}}$ the overall quantity of objects o delivered to b is at least q .
- for each $(w, l) \in store^{\mathbb{D}}$ the overall quantity of objects stored in w is at most l .

By answering the query Q on the database \mathbb{D} and introducing a linear program variable θ_Q^α for each of the query answer α , we can interpret the $LP(CQ_{\Sigma})$ program in Figure 1 as a linear program. A solution to this linear program will associate a real weight to each α , that is, to each tuple (f, w, b, o) that is a solution of Q over \mathbb{D} . Intuitively, this weight is the quantity of object o that factory f has to produce to store in warehouse w before being sent to buyer b . Moreover, these quantities are compatible with the constraints imposed on the capacity of each factory, warehouse and on the orders of each buyer. Hence an optimal solution of this linear program will yield an optimal way of producing what is necessary while minimizing the transportation costs.

¹In this constraint, we use the construct $\mathbf{num}(q)$ to explicitly specify that the domain of q is \mathbb{R}^+ and that, when evaluating the $LP(CQ_{\Sigma})$ program on a database \mathbb{D} , q will be decoded back as an element of \mathbb{R}^+ .

1.2. Related Work. Our result builds on well-known techniques using dynamic programming on tree decompositions of the hypergraph of conjunctive queries. These techniques were first introduced by Yannkakis [Yan81] who observed that so-called acyclic conjunctive queries could be answered in linear time using dynamic programming on a tree whose nodes are in correspondence with the atoms of the query. Generalizations have followed in two directions: on the one hand, generalizations of acyclicity such as notions of hypertree width [GLS02, GLS99, Gro06] have been introduced and on the other hand enumeration and aggregation problems have been shown to be tractable on these families of queries such as finding the size of the answer set [PS13] or enumerating it with small delay [BDG07]. These tractability results can be obtained in a unified and generalized way by using factorized databases introduced by Olteanu and Závodný [OZ12, OZ15], from which our work is inspired. Factorized databases provide succinct representations for answer sets of queries on databases. The representation enjoys interesting syntactic properties allowing to efficiently solve numerous aggregation problems on answer sets in polynomial time in the size of the representation. Olteanu and Závodný [OZ15] have shown that when the fractional hypertree width of a query Q is bounded, then one can construct, given a hypertree decomposition of Q and a database \mathbb{D} , a factorized database representing the answers of Q on \mathbb{D} of polynomial size. They also give a $O(1)$ delay enumeration algorithm on factorized databases. Combining both results gives a generalization of the result of Bagan, Durand and Grandjean [BDG07] on the complexity of enumerating the answers of conjunctive queries.

Our result heavily draws inspiration from this approach as we use bottom up dynamic programming on hypertree decomposition of the input query Q to construct a partial representation of the answers set of Q on database \mathbb{D} that we later use to construct a factorized interpretation of the linear program to solve. While our approach could be made to work directly on factorized representations of queries answer sets as defined by Olteanu and Závodný [OZ15], we choose to directly work on tree decompositions. One reason for this is because our factorized interpretation uses hypertree decompositions that are slightly more constraint than the one usually used to efficiently handle complex linear programs. Namely, our tree decomposition needs extra bags for dependencies between variables that are not present in the query but only in the linear program. This constraints are not straightforward to translate into factorized databases while they are natural on tree decompositions.

Comparison with conference version. This paper is a longer version of [CCNR22]. We have improved the presentation of some results from this old version, added new ones and added the full proofs left in the appendix in the earlier version. Some clarifications were made through slight changes in the theoretical framework that are described in the next paragraph. Our new contributions includes:

- A precise complexity analysis on how one can solve linear programs in $LP(CQ_{\Sigma})$ depending on their structure, stated by explicitly using the AGM bound and fractional hypertree width of the queries involved in the linear program,
- New hardness results for the general case,
- A cleaner logical framework to describe linear programs over database queries.

The main change in the presentation comes from the introduction of the core language $LP_{clos}(CQ_{\Sigma})$ on which the factorized interpretation is described. It allows for a cleaner analysis of the complexity of our approach where we can separate the explanation of interpreting $LP_{clos}(CQ_{\Sigma})$ languages, now called the *natural interpretation* (*naive interpretation* in the conference paper), and the unfolding of quantifiers in $LP(CQ_{\Sigma})$. Indeed, in the

conference version, we started by unfolding quantifiers before performing the analysis. This unfolding is now made explicit by the closure operation of $LP(CQ_\Sigma)$ programs which produces an $LP_{clos}(CQ_\Sigma)$ program that can then be solved using our techniques. It allows us to properly separate the unfolding phase from the interpretation phase and to describe their complexity independently. In particular, in the conference version of the paper, the tractability result holds only for a fragment of $LP(CQ_\Sigma)$ where we are able to bound the size of the unfolding. Thanks to the introduction of $LP_{clos}(CQ_\Sigma)$ and to a notion of normal form for $LP(CQ_\Sigma)$ programs, we are now able to state precise complexity bounds for every program in $LP(CQ_\Sigma)$ depending on their structure without assuming anything more. We also removed one constructor from the definition of $LP(CQ_\Sigma)$. Indeed, in the conference version of the paper, $LP(CQ_\Sigma)$ programs could use an expression of the form $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ to generate a linear sum depending on both Q and Q' . However, our tractability results worked only when Q' has a very particular form, namely, Q' needed to be of the form $\mathbf{x}=\mathbf{y}$. We hence removed this constructor from the definition of $LP_{clos}(CQ_\Sigma)$ language which has now only constructors of the form $\mathbf{weight}_{\mathbf{x}=\mathbf{c}}(Q)$ for a vector of database constants \mathbf{c} . Consequently we do not need to introduce a fragment of the general language anymore to recover tractability since the tractable case is now the only one possible in $LP_{clos}(CQ_\Sigma)$. It may appear that we lost some expressivity along the way but it turns out that we can recover the same behavior using constructors in $LP(CQ_\Sigma)$. Namely $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ can now be expressed as $\sum_{\mathbf{y}:Q'} \mathbf{weight}_{\mathbf{x}=\mathbf{y}}(Q)$.

1.3. Organization of the paper. Section 2 contains the necessary definitions to understand the paper. Section 3 presents the language $LP_{clos}(CQ_\Sigma)$ of linear programs parameterized by conjunctive queries and gives its semantics by interpreting programs in $LP_{clos}(CQ_\Sigma)$ as linear programs, which we call the natural interpretation. This language is very simple and does not allow universal quantification as used in Section 1.1. We show in Section 4 that one can exploit hypertree decomposition to compute the optimal value of $LP_{clos}(CQ_\Sigma)$ programs efficiently by interpreting them as more succinct linear programs, via an interpretation that we call factorized interpretation. The soundness of this approach is delayed to Section 6 as it contains results on weightings of conjunctive queries that are of independent interest. We then proceed to defining the language $LP(CQ_\Sigma)$ in Section 5.1. The language is more expressive than $LP_{clos}(CQ_\Sigma)$ as it allows for universal quantification over the database, as it is hinted in the previous example. We give its semantics via a closure operation that transforms an $LP(CQ_\Sigma)$ program to an $LP_{clos}(CQ_\Sigma)$ program. We analyze the complexity of solving $LP(CQ_\Sigma)$ programs and show how one can leverage the results on $LP_{clos}(CQ_\Sigma)$ to this program in Section 5.2. We present some preliminary experimental results in Section 5.3. Finally, Section 7 presents some applications of $LP(CQ_\Sigma)$.

2. PRELIMINARIES

Sets, Functions and Relations. Let $\mathbb{B} = \{0, 1\}$ be the set of Booleans, \mathbb{N} the set of natural numbers including 0, \mathbb{R}^+ be the set of positive reals including 0 and subsuming \mathbb{N} , and \mathbb{R} the set of all reals.

Given any set S and $n \in \mathbb{N}$ we denote by S^n the set of all n -tuples over S and by $S^* = \cup_{n \in \mathbb{N}} S^n$ the set of all words over S . A *weighting* on S is a (total) function $f : S \rightarrow \mathbb{R}^+$.

Given a set of (total) functions $A \subseteq \mathbb{D}^S = \{f \mid f : S \rightarrow \mathbb{D}\}$ and a subset $S' \subseteq S$, we define the set of restrictions $A|_{S'} = \{f|_{S'} \mid f \in A\}$. For any binary relation $R \subseteq S \times S$,

we denote its transitive closure by $R^+ \subseteq S \times S$ and the reflexive transitive closure by $R^* = R^+ \cup \{(s, s) \mid s \in S\}$.

Variable assignments. We fix a countably infinite set of (query) variables \mathcal{X} . For any set D of database elements, an assignment of (query) variables to database elements is a function $\alpha : X \rightarrow D$ that maps elements of a finite subset of variables $X \subseteq \mathcal{X}$ to values of D . For any two sets of variable assignments $A_1 \subseteq D^{X_1}$ and $A_2 \subseteq D^{X_2}$ we define their join $A_1 \bowtie A_2 = \{\alpha_1 \cup \alpha_2 \mid \alpha_1 \in A_1, \alpha_2 \in A_2, \alpha_1|_I = \alpha_2|_I\}$ where $I = X_1 \cap X_2$.

We also use a few vector notations. Given a vector of variables $\mathbf{x} = (x_1, \dots, x_n) \in \mathcal{X}^n$ we denote by $set(\mathbf{x}) = \{x_1, \dots, x_n\}$ the set of the elements of \mathbf{x} . For any variable assignment $\alpha : X \rightarrow D$ with $set(\mathbf{x}) \subseteq X$ we denote the application of the assignment α on \mathbf{x} by $\alpha(\mathbf{x}) = (\alpha(x_1), \dots, \alpha(x_n))$.

| | | | |
|--------------------|----------------|-------|---|
| Linear expressions | $S, S' \in LE$ | $::=$ | $r \mid \xi \mid rS \mid S + S'$ |
| Linear constraints | $C, C' \in LC$ | $::=$ | $S \leq S' \mid C \wedge C' \mid true$ |
| Linear programs | $L \in LP$ | $::=$ | maximize S subject to C |

Figure 2: The set of linear programs LP with variables $\xi \in \Xi$ and constants $r \in \mathbb{R}$.

2.1. Linear Programs. Let Ξ be a set of linear program variables. In Figure 2, we recall the definition of the sets of linear expressions LE , linear constraints LC , and linear programs LP with variables in Ξ . We consider the usual linear equations $S = S'$ as syntactic sugar for the constraints $S \leq S' \wedge S' \leq S$. For any linear program

$$L = \mathbf{maximize} \ S \ \mathbf{subject\ to} \ C$$

we call S the objective function of L and C the constraint of L . Note that the linear program **minimize** S **subject to** C can be expressed by

$$\mathbf{maximize} \ -1 \cdot S \ \mathbf{subject\ to} \ C.$$

We recall the formal semantics of linear programs in Figure 3.

$$\begin{aligned}
 eval_w(r) &= r && \text{with } r \in \mathbb{R}, w : \Xi \rightarrow \mathbb{R} \\
 eval_w(\xi) &= w(\xi) && \text{with } \xi \in \Xi \\
 eval_w(rS) &= r \cdot eval_w(S) \\
 eval_w(S + S') &= eval_w(S) + eval_w(S') \\
 \llbracket true \rrbracket &= \{w \mid w : \Xi \rightarrow \mathbb{R}^+\} \\
 \llbracket S \leq S' \rrbracket &= \{w \mid eval_w(S) \leq eval_w(S')\} \\
 \llbracket C \wedge C' \rrbracket &= \llbracket C \rrbracket \cap \llbracket C' \rrbracket \\
 opt(\mathbf{maximize} \ S \ \mathbf{subject\ to} \ C) &= \max(\{eval_w(S) \mid w : \Xi \rightarrow \mathbb{R}^+, w \in \llbracket C \rrbracket\})
 \end{aligned}$$

Figure 3: Semantics of linear expressions, constraints and programs.

For any weightings $w : \Xi \rightarrow \mathbb{R}^+$, the value of a sum $S \in LE$ is the real number $eval_w(S) \in \mathbb{R}$. We denote the solution set of a constraint $C \in LC$ by $\llbracket C \rrbracket \subseteq \{w \mid w : \Xi \rightarrow \mathbb{R}^+\}$.

The optimal value $opt(L) \in \mathbb{R}$ of a linear program L with objective function S and constraint C is:

$$opt(L) = \max\{eval_w(S) \mid w : \Xi \rightarrow \mathbb{R}^+, w \in \llbracket C \rrbracket\}$$

The *size* $|L|$ of a linear program L is defined to be the number of symbols needed to write it down. It is well-known that the optimal solution of a linear program L can be computed in polynomial time in $|L|$ [Kar84].

Observe that we are only interested in non-negative weightings, without explicitly imposing positivity constraints. It is a usual assumption in linear programming since it is well known that one can transform any linear program L into L' of size at most $2|L|$ so that the feasible points of L' over \mathbb{R}^+ are exactly the feasible points of L over \mathbb{R} , by simply replacing every occurrence of a variable x in L by $x^+ - x^-$.

2.2. Conjunctive Queries. A *relational signature* is a pair $\Sigma = (\mathcal{R}, \mathcal{C})$ where \mathcal{C} a finite set of constants ranged over by c and $\mathcal{R} = \cup_{n \in \mathbb{N}} \mathcal{R}^{(n)}$ is a finite set of relation symbols. The elements $R \in \mathcal{R}^{(n)}$ are called relation symbols of arity $n \in \mathbb{N}$.

$$\begin{array}{l} \text{Expressions} \quad E_1, \dots, E_n \quad ::= x \mid c \\ \text{Conjunctive queries } Q, Q' \in CQ_\Sigma \quad ::= E_1 \doteq E_2 \mid R(E_1, \dots, E_n) \\ \quad \quad \quad \quad \quad \quad \quad \quad \mid Q \wedge Q' \mid \exists x.Q \mid true \end{array}$$

Figure 4: The set of conjunctive queries CQ_Σ with signature $\Sigma = ((\mathcal{R}^{(n)})_{n \in \mathbb{N}}, \mathcal{C})$ where $x \in \mathcal{X}$, $c \in \mathcal{C}$, and $R \in \mathcal{R}^{(n)}$.

In Figure 4 we recall the notion of conjunctive queries. An expression E is either a (query) variable $x \in \mathcal{X}$ or a constant $a \in \mathcal{C}$. The set of conjunctive queries $Q \in CQ_\Sigma$ is built from equations $E_1 \doteq E_2$, atoms $R(E_1, \dots, E_n)$, the logical operators of conjunction $Q \wedge Q'$ and existential quantification $\exists x.Q$. Given a vector $\mathbf{x} = (x_1, \dots, x_n) \in \mathcal{X}^n$ and a query Q , we write $\exists \mathbf{x}.Q$ instead of $\exists x_1. \dots \exists x_n.Q$. For any sequence of constants $\mathbf{c} = (c_1, \dots, c_n) \in \mathcal{C}^n$ we write $\mathbf{x} \doteq \mathbf{c}$ instead of $x_1 \doteq c_1 \wedge \dots \wedge x_n \doteq c_n$. If $n = 0$ then $\mathbf{x} \doteq \mathbf{c}$ is equal to *true*.

The set of free variables $fv(Q) \subseteq \mathcal{X}$ are those variables that occur in Q outside the scope of an existential quantifier:

$$\begin{array}{ll} fv(R(E_1, \dots, E_n)) = \bigcup_{i=1}^n fv(E_i) & fv(E_1 \doteq E_2) = fv(E_1) \cup fv(E_2) \\ fv(Q \wedge Q') = fv(Q) \cup fv(Q') & fv(\exists x.Q) = fv(Q) \setminus \{x\} \\ fv(x) = \{x\} & fv(c) = \emptyset \end{array}$$

A conjunctive query Q is said to be *quantifier free* if it does not contain any existential quantifier. In the literature, such queries are sometimes also called full queries.

We can define operations to extend queries with additional variables \mathbf{x} such that for all $Q \in CQ_\Sigma$:

$$ext_{\mathbf{x}}(Q) = \bigwedge_{x \in \text{set}(\mathbf{x}) \setminus fv(Q)} x \doteq x \wedge Q$$

For any $n \geq 0$ and vector of constants $\mathbf{c} \in \mathcal{C}^n$ and vector of variables $\mathbf{x} \in \mathcal{X}^n$ we define an operator $subs_{[\mathbf{x}/\mathbf{c}]}$ on conjunctive queries, that substitutes any variable in a vector \mathbf{x} by the constant at the same position in vector \mathbf{c} , so that for all queries $Q \in CQ_\Sigma$:

$$subs_{[\mathbf{x}/\mathbf{c}]}(Q) = Q[\mathbf{x}/\mathbf{c}],$$

i.e., where all occurrences in Q of variables in \mathbf{x} are replaced by the corresponding elements of \mathbf{c} .

2.3. Relational Databases. A relational Σ -structure is a tuple $\mathbb{D} = (\Sigma, D, \cdot^{\mathbb{D}})$, where $\Sigma = ((\mathcal{R}^{(n)})_{n \geq 0}, \mathcal{C})$ is a relational signature, D a finite set, $c^{\mathbb{D}} \in D$ an element for each constant $c \in \mathcal{C}$ and $R^{\mathbb{D}} \subseteq D^n$ a relation for any relation symbol $R \in \mathcal{R}^{(n)}$ and $n \geq 0$. We also define the structures' domain $\text{dom}(\mathbb{D}) = D$. A (*relational*) *database* \mathbb{D} is a finite relational Σ -structure, i.e., all its components are finite. We denote the set of all databases by db_{Σ} .

For any conjunctive query $Q \in CQ_{\Sigma}$, set $X \supseteq \text{fv}(Q)$ and relational database $\mathbb{D} \in db_{\Sigma}$ we define the answer set $\llbracket Q \rrbracket_X^{\mathbb{D}}$ in Figure 5. It contains all those assignments $\alpha : X \rightarrow D$ for which Q becomes true on \mathbb{D} .

$$\begin{aligned}
 \text{eval}^{\mathbb{D}, \alpha}(x) &= \alpha(x) && \text{(with } x \in \mathcal{X} \text{ a variable)} \\
 \text{eval}^{\mathbb{D}, \alpha}(c) &= c^{\mathbb{D}} && \text{(with } c \in \mathcal{C} \text{ a constant)} \\
 \llbracket E_1 \doteq E_2 \rrbracket_X^{\mathbb{D}} &= \{\alpha : X \rightarrow D \mid \text{eval}^{\mathbb{D}, \alpha}(E_1) = \text{eval}^{\mathbb{D}, \alpha}(E_2)\} \\
 \llbracket R(E_1, \dots, E_n) \rrbracket_X^{\mathbb{D}} &= \{\alpha : X \rightarrow D \mid (\text{eval}^{\mathbb{D}, \alpha}(E_1), \dots, \text{eval}^{\mathbb{D}, \alpha}(E_n)) \in R^{\mathbb{D}}\} \\
 \llbracket Q_1 \wedge Q_2 \rrbracket_X^{\mathbb{D}} &= \llbracket Q_1 \rrbracket_X^{\mathbb{D}} \cap \llbracket Q_2 \rrbracket_X^{\mathbb{D}} \\
 \llbracket \exists x. Q \rrbracket_X^{\mathbb{D}} &= \{\alpha|_X \mid \alpha \in \llbracket Q \rrbracket_{X \cup \{x\}}^{\mathbb{D}}\} && \text{if } x \notin X \\
 \llbracket \text{true} \rrbracket_X^{\mathbb{D}} &= X^D
 \end{aligned}$$

Figure 5: The answer set of a conjunctive query $Q \in CQ_{\Sigma}$ on a database $\mathbb{D} \in db_{\Sigma}$ for a set of variables $X \supseteq \text{fv}(Q)$.

We define the semantics of a query by:

$$\llbracket Q \rrbracket^{\mathbb{D}} = \llbracket Q \rrbracket_{\text{fv}(Q)}^{\mathbb{D}}$$

In particular, observe that the semantics of existential quantifiers is the projection of the answer set, that is: $\llbracket \exists \mathbf{x}. Q \rrbracket^{\mathbb{D}} = \llbracket Q \rrbracket_{\text{fv}(Q) \setminus \text{set}(\mathbf{x})}^{\mathbb{D}}$.

2.4. Hypertree Decompositions. Hypertree decompositions of conjunctive queries are a way of laying out the structure of a conjunctive query in a tree. It allows to solve many aggregation problems (such as checking the existence of a solution, counting or enumerating the solutions etc.) on quantifier free conjunctive queries in polynomial time where the degree of the polynomial is given by the width of the decomposition.

A digraph is a pair $(\mathcal{V}, \mathcal{E})$ with node set \mathcal{V} and edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. A digraph is acyclic if there is no $v \in \mathcal{V}$ for which $(v, v) \in \mathcal{E}^+$. For any node $u \in \mathcal{V}$, we denote by $\downarrow u = \{v \in \mathcal{V} \mid (u, v) \in \mathcal{E}^*\}$ the set of nodes in \mathcal{V} reachable over some downwards path from u , and we define the context of u , denoted $\uparrow u$, by $\uparrow u = (\mathcal{V} \setminus \downarrow u) \cup \{u\}$. The digraph $(\mathcal{V}, \mathcal{E})$ is a forest if it is acyclic and for all $u, u', v \in \mathcal{V}$ there holds that $(u, v), (u', v) \in \mathcal{E}$ implies $u = u'$. Moreover, $(\mathcal{V}, \mathcal{E})$ is a tree if there exists a node $r \in \mathcal{V}$ such that $\mathcal{V} = \downarrow r$. In this case, r is unique and called the root of the tree. If for $v \in \mathcal{V}$ it holds that $\downarrow v = \{v\}$, then v is called a leaf. Observe that in this tree, the paths are oriented from the root to the leaves of the tree.

Definition 2.1. Let $X \subseteq \mathcal{X}$ be a finite set of variables. A decomposition tree T of X is a tuple $(\mathcal{V}, \mathcal{E}, \mathcal{B})$ such that:

- $(\mathcal{V}, \mathcal{E})$ is a finite directed rooted tree with edges from the root to the leaves,
- the bag function $\mathcal{B} : \mathcal{V} \rightarrow 2^X$ maps nodes to subsets of variables in X ,
- for all $x \in X$ the subset of nodes $\{u \in \mathcal{V} \mid x \in \mathcal{B}(u)\}$ is connected in the tree $(\mathcal{V}, \mathcal{E})$,
- each variable of X appears in some bag, that is $\bigcup_{u \in \mathcal{V}} \mathcal{B}(u) = X$.

Now a hypertree decomposition of a quantifier free conjunctive query is a decomposition tree where for each atom of the query there is at least one bag that covers its variables.

Definition 2.2 (Hypertree width of quantifier free conjunctive queries). Let $Q \in CQ_\Sigma$ be a quantifier free conjunctive query. A *generalized hypertree decomposition* of Q is a decomposition tree $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ of $fv(Q)$ such that for each atom $R(\mathbf{x})$ of Q there is a vertex $u \in \mathcal{V}$ such that $set(\mathbf{x}) \subseteq \mathcal{B}(u)$. The *width* of T with respect to Q is the minimal number k such that every bag of T can be covered by the variables of k atoms of Q . The *generalized hypertree width of a query Q* is the minimal width of a tree decomposition of Q .

For example, the query $R(x, y) \wedge R(y, z)$ has a generalized hypertree decomposition of hypertree width 1: $(\mathcal{V}, \mathcal{E}, \mathcal{B})$ with vertices $\mathcal{V} = \{1, 2, 3\}$, edges $\mathcal{E} = \{(1, 2), (1, 3)\}$, and bags $\mathcal{B} = [1/\{y\}, 2/\{x, y\}, 3/\{y, z\}]$.

While hypertree width allows to obtain efficient algorithms on conjunctive queries, our results will also work for the more general notion of *fractional hypertree width*, which consists in a fractional relaxation of the hypertree width. We let $Q \in CQ_\Sigma$ be a quantifier free conjunctive query, A be the atoms of Q and let $X \subseteq fv(Q)$. A *fractional cover* of X is a function $c : A \rightarrow \mathbb{R}^+$ assigning positive weights to the atoms of Q such that for every $x \in X$, $\sum_{R \in A, x \in fv(R)} c(R) \geq 1$. The *value of a fractional cover c* is defined as $\sum_{R \in A} c(R)$.

For example, consider the query $Triangle = R(x, y) \wedge S(y, z) \wedge T(z, x)$ and $X = \{x, y, z\}$. The function c such that $c(R) = c(S) = c(T) = 1/2$ is a fractional cover of X of value $3/2$.

Definition 2.3. Let Q be a conjunctive query and $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a generalized hypertree decomposition of Q . The *fractional hypertree width* of T is the smallest k such that for every $u \in \mathcal{V}$, there exists a fractional cover of $\mathcal{B}(u)$ of value smaller than k . The *fractional hypertree width* of Q , denoted by $fhtw(Q)$, is the smallest k such that Q has a generalized hypertree decomposition of fractional hypertree width k .

From now on, we will only write the width of T in place of the fractional hypertree width. The key observation making fractional hypertree width suitable for algorithmic purposes is due to Grohe and Marx [GM14] who proved that if a quantifier free conjunctive query is such that $fv(Q)$ has a fractional cover of value k , then $|\llbracket Q \rrbracket^\mathbb{D}| \leq |\mathbb{D}|^k$. Hence, if $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ is a tree decomposition of Q of width k , then $\llbracket Q \rrbracket_{\mathcal{B}(u)}^\mathbb{D}$ is of size at most $|\mathbb{D}|^k$. Moreover, it can be computed efficiently:

Lemma 2.4. *Given a tree decomposition $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ of a quantifier free conjunctive query $Q \in CQ_\Sigma$ of width k and a database $\mathbb{D} \in db_\Sigma$, one can compute the collection of bag projections $(\llbracket Q \rrbracket_{\mathcal{B}(u)}^\mathbb{D})_{u \in \mathcal{V}}$ in time $O((|\mathbb{D}|^k \log(|\mathbb{D}|)) \cdot |\mathcal{V}|)$. Moreover, for every $u \in \mathcal{V}$, $\llbracket Q \rrbracket_{\mathcal{B}(u)}^\mathbb{D}$ is of size at most $|\mathbb{D}|^k$.*

Lemma 2.4 is folklore: it can be proven by computing the semi-join of every bag in a subtree in a bottom-up fashion, as it is done in [Lib13, Theorem 6.25] and using a worst-case optimal join algorithm such as Triejoin [Vel14] for computing the relation at each bag. This yields a superset S_u of $\llbracket Q \rrbracket_{\mathcal{B}(u)}^\mathbb{D}$ for every u . Then, with a second top-down phase, one can remove tuples from S_u that cannot be extended to a solution of $\llbracket Q \rrbracket^\mathbb{D}$.

Following the previously mentioned upper bound of [GM14], Atserias, Grohe and Marx proved in [AGM13] that the bound given by an optimal fractional cover of Q is tight (up to polynomial factors). This bound is now usually referred to as the AGM bound. More precisely, it says that if $\text{AGM}(Q)$ denotes the smallest value over every fractional cover of Q , then for every \mathbb{D} , $\llbracket Q \rrbracket^{\mathbb{D}}$ is of size at most $|\mathbb{D}|^{\text{AGM}(Q)}$ and there exists a database \mathbb{D}^* such that $\llbracket Q \rrbracket^{\mathbb{D}^*}$ is of size greater than $\frac{|\mathbb{D}^*|^{\text{AGM}(Q)}}{\text{poly}(|Q|)}$. Hence, even if Q is of width k , the size of $\llbracket Q \rrbracket^{\mathbb{D}}$ could be order of magnitudes bigger than $|\mathbb{D}|^k$ when $k < \text{AGM}(Q)$. Hence, Lemma 2.4 gives a succinct way of describing the set of solutions of Q that we exploit in this paper.

Parts of our result will be easier to describe on so-called normalized decomposition trees:

Definition 2.5. Let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a decomposition tree. We call a node $u \in \mathcal{V}$ of T :

- **an extend node:** if it has a single child u' and $\mathcal{B}(u) = \mathcal{B}(u') \cup \{x\}$ for some $x \in \mathcal{X} \setminus \mathcal{B}(u')$,
- **a project node:** if it has a single child u' and $\mathcal{B}(u) = \mathcal{B}(u') \setminus \{x\}$ for some $x \in \mathcal{X} \setminus \mathcal{B}(u)$,
- **a join node:** if it has $k \geq 1$ children u_1, \dots, u_k with $\mathcal{B}(u) = \mathcal{B}(u_1) = \dots = \mathcal{B}(u_k)$.

We call T *normalized*² if all its nodes in \mathcal{V} are either extend nodes, project nodes, join nodes, or leaves.

It is well-known that tree decompositions can always be normalized without changing the width. Thus normalization does not change the asymptotic complexity of the algorithms.

Lemma 2.6 (Lemma of 13.1.2 of [Klo94]). *For every tree decomposition of $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ of Q of width k , there exists a normalized tree decomposition $T' = (\mathcal{V}', \mathcal{E}', \mathcal{B}')$ having width k . Moreover, one can compute T' from T in polynomial time.*

3. LINEAR PROGRAMS WITH CLOSED WEIGHT EXPRESSIONS

In this section, we introduce the language $LP_{\text{clos}}(CQ_{\Sigma})$ to express linear programs parameterized by conjunctive queries. This language is deliberately kept simple, which allow us to design efficient algorithms for it. An element of $LP_{\text{clos}}(CQ_{\Sigma})$ is called a *closed LP* (CQ_{Σ}) *program*. We refer to such programs as “closed” because they do not contain quantification in the linear program part, which contrasts with the more general definition of $LP(CQ_{\Sigma})$ given in Section 5, which allows to express more interesting linear program. The case of closed $LP(CQ_{\Sigma})$ programs is however central in this work as this is the class of optimization problems for which we propose an efficient algorithm. The more general case of $LP(CQ_{\Sigma})$ programs is dealt with using a “closure” procedure which transforms any programs from $LP(CQ_{\Sigma})$ into a closed program $LP_{\text{clos}}(CQ_{\Sigma})$.

Let Σ be a relational signature. A *closed weight expression* is an expression of the form $\mathbf{weight}_{\mathbf{x}=\mathbf{c}}(Q)$ where Q is a conjunctive query, $\text{set}(\mathbf{x}) \subseteq \text{fv}(Q)$, variables in \mathbf{x} are pairwise distinct and \mathbf{c} are database constants. An $LP_{\text{clos}}(CQ_{\Sigma})$ program is intuitively a linear program whose variables are closed weight expressions. A formal definition is given in Figure 6.

²In the literature this property is referred to as “nice” tree decompositions.

| | | | |
|--------------------|----------------------------------|-------|--|
| Linear sums | $S, S' \in LS_{clos}(CQ_\Sigma)$ | $::=$ | weight $_{\mathbf{x} \doteq \mathbf{c}}(Q) \mid rS \mid S + S' \mid r$ where $set(\mathbf{x}) \subseteq fv(Q)$, variables in \mathbf{x} are pairwise distinct \mathbf{c} are database constants |
| Linear constraints | $C, C' \in LC_{clos}(CQ_\Sigma)$ | $::=$ | $S \leq S' \mid S \doteq S' \mid C \wedge C' \mid true$ |
| Linear programs | $L \in LP_{clos}(CQ_\Sigma)$ | $::=$ | maximize S subject to C |

Figure 6: Linear sums, constraints, and programs with closed weight expressions containing conjunctive queries $Q \in CQ_\Sigma$ where $r \in \mathbb{R}$.

Such linear programs can be interpreted as standard linear programs for any database \mathbb{D} with numerical values. In order to do so, we fix for any query $Q \in CQ_\Sigma$ a set $\Theta_Q^\mathbb{D}$ of fresh linear program variables θ_Q^α arbitrarily:

$$\Theta_Q^\mathbb{D} = \{\theta_Q^\alpha \mid \alpha \in \llbracket Q \rrbracket^\mathbb{D}\}$$

We can then map each closed weight expression to a linear sum with variables in $\Theta_Q^\mathbb{D}$ as follows:

$$\langle \mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q) \rangle^\mathbb{D} = \sum_{\substack{\alpha \in \llbracket Q \rrbracket^\mathbb{D} \\ \alpha(\mathbf{x}) = \mathbf{c}}} \theta_Q^\alpha$$

Note that our assumption $set(\mathbf{x}) \subseteq fv(Q)$ ensures that $\alpha(\mathbf{x})$ is well-defined.

Definition 3.1. For any linear program $L \in LP_{clos}(CQ_\Sigma)$ we define the *natural interpretation* $\langle L \rangle^\mathbb{D} \in LP$ by replacing any weight expression S in L by $\langle S \rangle^\mathbb{D}$. By applying the same substitution we define the interpretation $\langle S \rangle^\mathbb{D} \in LS$ of any linear sum $S \in LS_{clos}(CQ_\Sigma)$ and the interpretation $\langle C \rangle^\mathbb{D} \in LC$ of any linear constraint $C \in LC_{clos}(CQ_\Sigma)$ in analogy.

The *size* $|L|$ of a program $L \in LP_{clos}(CQ_\Sigma)$ is defined to be the number of symbols needed to write it down.

3.1. Example. As an example we consider the conjunctive query $Q = R_1(x) \wedge R_2(y)$ and the following program $L \in LP_{clos}(CQ_\Sigma)$:

$$\begin{array}{ll} \mathbf{maximize} & \mathbf{weight}_\emptyset(Q) \\ \mathbf{subject\ to} & \mathbf{weight}_{x \doteq 0}(Q) \leq 1 \\ & \wedge \mathbf{weight}_{x \doteq 1}(Q) \leq 1 \end{array}$$

Let \mathbb{D} be the database \mathbb{D} with tables $R_1^\mathbb{D} = \{(0), (1)\}$ and $R_2^\mathbb{D} = \{(0), (1)\}$. The answer set of Q is $\llbracket Q \rrbracket^\mathbb{D} = \{\alpha \mid \alpha : \{x, y\} \rightarrow \{0, 1\}\}$. The interpretation $\langle L \rangle^\mathbb{D}$ is the following linear program, where we denote any query answer $\alpha \in \llbracket Q \rrbracket^\mathbb{D}$ by a pair $(\alpha(x), \alpha(y))$ in the Cartesian product $\{0, 1\}^2$ for simplicity:

$$\begin{array}{ll} \mathbf{maximize} & \theta_Q^{(0,0)} + \theta_Q^{(0,1)} + \theta_Q^{(1,0)} + \theta_Q^{(1,1)} \\ \mathbf{subject\ to} & \theta_Q^{(0,0)} + \theta_Q^{(0,1)} \leq 1 \\ & \wedge \theta_Q^{(1,0)} + \theta_Q^{(1,1)} \leq 1 \end{array}$$

The objective function $\theta_Q^{(0,0)} + \theta_Q^{(0,1)} + \theta_Q^{(1,0)} + \theta_Q^{(1,1)}$ is the interpretation of the expression $\mathbf{weight}_\emptyset(Q)$. The first constraint $\theta_Q^{(0,0)} + \theta_Q^{(0,1)}$ is obtained by interpreting $\mathbf{weight}_{x \doteq 0}(Q) \leq 1$

and the second constraint by interpreting $\mathbf{weight}_{x \doteq 1}(Q) \leq 1$. Note that the objective function is the sum of the lefthandsides of the two constraints, so the three weight expressions of L are semantically related.

3.2. Complexity of solving $LP_{clos}(CQ_\Sigma)$. In this section, we are interested in the complexity of computing $opt(\langle L \rangle^\mathbb{D})$ given $L \in LP_{clos}(CQ_\Sigma)$ and a database \mathbb{D} . From a combined complexity point of view, that is, when both L and \mathbb{D} are assumed to be part of the input, it is not hard to see that the problem is NP-hard since it requires to implicitly find the answer set of every conjunctive query appearing in L . We formalize this intuition in the following theorem:

Theorem 3.2. *The problem of deciding whether $opt(\langle L \rangle^\mathbb{D}) \neq 0$ given a relational signature Σ , $L \in LP_{clos}(CQ_\Sigma)$ and a database \mathbb{D} is both NP-hard and coNP-hard.*

Proof. It is well-known that the problem of deciding whether $\llbracket Q \rrbracket^\mathbb{D} \neq \emptyset$ given a conjunctive query Q and a database \mathbb{D} in the input is NP-complete [CM77]. We show that this problem can be reduced to the problem of deciding whether the optimal value $opt(\langle L \rangle^\mathbb{D})$ is non-zero, given a relational signature Σ , a linear program $L \in LP_{clos}(CQ_\Sigma)$ and a database \mathbb{D} with schema Σ . The NP-hardness of computing $opt(\langle L \rangle^\mathbb{D})$ is thus a direct corollary.

For any conjunctive query Q , we consider the following $LP_{clos}(CQ_\Sigma)$ program:

$$L_Q = \mathbf{maximize} \mathbf{weight}_{true}(Q) \mathbf{subject\ to} \mathbf{weight}_{true}(Q) \leq 1$$

We claim that

$$opt(\langle L_Q \rangle^\mathbb{D}) \neq 0 \text{ if and only if } \llbracket Q \rrbracket^\mathbb{D} \neq \emptyset$$

We first note that:

$$\langle L_Q \rangle^\mathbb{D} = \mathbf{maximize} S \mathbf{subject\ to} S \leq 1 \quad \text{where } S = \sum_{\alpha \in \llbracket Q \rrbracket^\mathbb{D}} \theta_Q^\alpha$$

So, if $\llbracket Q \rrbracket^\mathbb{D} = \emptyset$, then $eval_w(S) = 0$ for all w , so that $opt(\langle L_Q \rangle^\mathbb{D}) = 0$. So consider the other case where $\llbracket Q \rrbracket^\mathbb{D} \neq \emptyset$. Let $\alpha \in \llbracket Q \rrbracket^\mathbb{D}$ and consider the weighting w such that $w(\theta_Q^\alpha) = 1$ and $w(\theta_Q^{\alpha'}) = 0$ for every $\alpha' \in \llbracket Q \rrbracket^\mathbb{D}$ such that $\alpha' \neq \alpha$. This weighting clearly respects the constraints of $S \leq 1$, so $w \in \llbracket S \leq 1 \rrbracket$ showing that $opt(\langle L_Q \rangle^\mathbb{D}) \geq 1 \neq 0$.

To show the coNP-hardness, it is sufficient to observe that the same trick can be applied to reduce the problem of deciding whether $\llbracket Q \rrbracket^\mathbb{D} = \emptyset$ given Q and \mathbb{D} , which is coNP-complete from [CM77]. \square

Data complexity. The hardness from Theorem 3.2 mainly stems from the hardness of answering conjunctive queries, that is only relevant in the context of combined complexity. It is often assumed however that the size of the query is small with respect to the size of the data, hence one can study the data complexity of the problem, that is, the complexity of the problem when we fix the linear program L . In this case, computing $opt(\langle L \rangle^\mathbb{D})$ can be done in polynomial time in $|\mathbb{D}|$ using the following procedure:

- Compute explicitly $\llbracket Q \rrbracket^\mathbb{D}$ for every Q appearing in L ,
- Compute $\langle L \rangle^\mathbb{D}$,
- Solve $\langle L \rangle^\mathbb{D}$ in time polynomial in $|\langle L \rangle^\mathbb{D}|$ using an LP-solver.

The exact complexity of the previous procedure is however dependent on the size of $\langle L \rangle^{\mathbb{D}}$ whose number of variables is the sum of $|\llbracket Q \rrbracket^{\mathbb{D}}|$ for every Q appearing in L . We lift the AGM bound presented in Section 2.4 to linear programs in $LP_{clos}(CQ_{\Sigma})$ by defining $\mathbf{AGM}(L)$ to be the maximum of $\mathbf{AGM}(Q)$ for every query Q appearing in L . The size of $\langle L \rangle^{\mathbb{D}}$ can now be upper bounded by $|L| \times |\mathbb{D}|^{\mathbf{AGM}(L)}$. Using a worst-case optimal join algorithm such as Triejoin [Vel14] to compute $\llbracket Q \rrbracket^{\mathbb{D}}$ in time $O(|\mathbb{D}|^{\mathbf{AGM}(Q)})$, we conclude that one can compute $opt(\langle L \rangle^{\mathbb{D}})$ in time $O((|L||\mathbb{D}|^{\mathbf{AGM}(L)})^{\ell})$, where ℓ is the best known constant to compute the optimal value of a linear program. Currently, the best known value for ℓ is smaller than 2.37286 by combining a result relating the complexity of solving linear programs with the complexity of multiplying matrices [CLS21] with the best known algorithm for multiplying matrices [AW21].

Theorem 3.3. *Given a relational signature Σ , $L \in LP_{clos}(CQ_{\Sigma})$ and a database \mathbb{D} , one can compute $opt(\langle L \rangle^{\mathbb{D}})$ in time $O(|L|^{\ell}|\mathbb{D}|^{\ell \cdot \mathbf{AGM}(L)})$ with $\ell < 2.37286$.*

3.3. Replacement Rewriting. In this paper, we will often introduce alternate ways of interpreting linear program of $LP_{clos}(CQ_{\Sigma})$ over a database. We will say that such alternate interpretation is *sound* if for any database \mathbb{D} , when interpreting the linear program using this alternate interpretation over \mathbb{D} , we obtain a linear program having the same optimal value as $\langle L \rangle^{\mathbb{D}}$, the natural interpretation of L over \mathbb{D} . It will allow us for example to construct smaller linear programs and hence speed up the computation of $opt(\langle L \rangle^{\mathbb{D}})$. Formally proving the soundness of an alternate interpretation is usually tedious as it involves transforming solutions from one interpretation to the other while proving by induction that the constraints in L are all satisfied. However, every interpretation that we will consider in this paper is based on interpreting weight expressions differently with some extra equality constraints. Hence, most of the time, the reasoning may be reduced to very simple linear programs involving only equality constraints. Our goal in this section is to provide formal tools to simplify soundness proofs.

One can always construct a function ν mapping every possible weight expression W into a fresh linear program variable $\nu(W)$ such that if $W \neq W'$, then $\nu(W) \neq \nu(W')$. We will often denote $\nu(\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q))$ by $\nu_Q^{\mathbf{x} \doteq \mathbf{c}}$. From now on, we assume such function ν has been fixed. For a set of weight expressions \mathbf{W} , we define the *weight constraints of \mathbf{W}* , denoted by $wc^{\nu}(\mathbf{W})$ as the following set of linear constraints:

$$\bigwedge_{W \in \mathbf{W}} \nu(W) = W.$$

For any linear sum $S \in LS_{clos}(CQ_{\Sigma})$, let $\langle S \rangle^{\nu} \in LS$ be defined by replacing any weight expression W in S by $\nu(W)$ and for $C \in LC_{clos}(CQ_{\Sigma})$, let $\langle C \rangle^{\nu} \in LP$ be the linear constraint obtained by applying the substitution to every linear sum appearing in C .

Consider a linear program $L = \mathbf{maximize} \ S \ \mathbf{subject\ to} \ C$ with some linear sum $S \in LS_{clos}(CQ_{\Sigma})$ and some linear constraint $C \in LC_{clos}(CQ_{\Sigma})$. We denote by $\mathbf{W}(L)$ the set of weight expressions that appear in L . The replacement rewriting of L is the following linear program $repl^{\nu}(L) \in LP(CQ_{\Sigma})$:

$$repl^{\nu}(L) = \mathbf{maximize} \ \langle S \rangle^{\nu} \ \mathbf{subject\ to} \ \langle C \rangle^{\nu} \wedge wc^{\nu}(\mathbf{W}(L)).$$

Observe that in $repl^{\nu}(L)$, the only place where $\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q)$ constructors appear is in the $wc^{\nu}(\mathbf{W}(L))$ part. Hence, we can naturally lift the interpretation of L over a database \mathbb{D}

to $\text{repl}^\nu(L)$ as follows:

$$\langle \text{repl}^\nu(L) \rangle^{\mathbb{D}} = \mathbf{maximize} \langle S \rangle^\nu \mathbf{subject\ to} \langle C \rangle^\nu \wedge \langle \text{wc}^\nu(\mathbf{W}(L)) \rangle^{\mathbb{D}}.$$

The main feature of $\text{repl}^\nu(L)$ is that it allows to formally separate the linear programming part from the part that is interpreted over a database, which will be helpful whenever we need to reason only on weight expressions.

Example. In the example of Section 3.1, the rewriting $\text{repl}^\nu(L)$ of L is:

$$\begin{aligned} \mathbf{maximize} \quad & \nu_2 \\ \mathbf{subject\ to} \quad & \nu_0 \leq 1 \wedge \\ & \nu_1 \leq 1 \wedge \\ & \nu_0 \doteq \mathbf{weight}_{x \doteq 0}(Q) \wedge \\ & \nu_1 \doteq \mathbf{weight}_{x \doteq 1}(Q) \wedge \\ & \nu_2 \doteq \mathbf{weight}_\emptyset(Q) \end{aligned}$$

which is then interpreted as $\langle \text{repl}^\nu(L) \rangle^{\mathbb{D}}$ as:

$$\begin{aligned} \mathbf{maximize} \quad & \nu_2 \\ \mathbf{subject\ to} \quad & \nu_0 \leq 1 \wedge \\ & \nu_1 \leq 1 \wedge \\ & \nu_0 \doteq \theta_Q^{(0,0)} + \theta_Q^{(0,1)} \wedge \\ & \nu_1 \doteq \theta_Q^{(1,0)} + \theta_Q^{(1,1)} \wedge \\ & \nu_2 \doteq \theta_Q^{(0,0)} + \theta_Q^{(0,1)} + \theta_Q^{(1,0)} + \theta_Q^{(1,1)} \end{aligned}$$

Note that the linear program's variables $\theta_Q^{(i,j)}$ do not occur in the objective function, so they are implicitly existentially quantified. Up to existential quantification, the above constraint is equivalent to:

$$\nu_2 \doteq \nu_0 + \nu_1$$

So we obtain a linear program with much fewer variables:

$$\mathbf{maximize} \nu_2 \mathbf{subject\ to} \nu_0 \leq 1 \wedge \nu_1 \leq 1 \wedge \nu_2 \doteq \nu_0 + \nu_1$$

The optimal value of this new linear program is achieved with the solution $\nu_0 = \nu_1 = 1$ and $\nu_2 = 2$. We can see that it directly corresponds to the optimal solution of the original linear program where $\theta_Q^{(0,0)} = \theta_Q^{(0,1)} = \theta_Q^{(1,0)} = \theta_Q^{(1,1)} = \frac{1}{2}$. How to derive such factorized rewritings of constraints and how to reconstruct an optimal solution from the optimal solution of the rewritten program from a given $LP(CQ_\Sigma)$ program in a systematic manner is studied in the remainder of this article.

It is easy to see that for every database \mathbb{D} over signature Σ , the optimal value of $\langle \text{repl}^\nu(L) \rangle^{\mathbb{D}}$ is the same as the optimal value of $\langle L \rangle^{\mathbb{D}}$ which is formalized as follows:

Proposition 3.4 (Soundness of replacement rewriting). *Given a database \mathbb{D} with signature Σ and a linear program $L \in LP_{\text{clos}}(CQ_\Sigma)$, we have:*

$$\text{opt}(\langle L \rangle^{\mathbb{D}}) = \text{opt}(\langle \text{repl}^\nu(L) \rangle^{\mathbb{D}}).$$

Proof. For every weight expression $\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q)$ of L and $w: \Theta_Q^{\mathbb{D}} \rightarrow \mathbb{R}^+$, we extend w to $\nu_Q^{\mathbf{x} \doteq \mathbf{c}}$ by defining

$$w(\nu_Q^{\mathbf{x} \doteq \mathbf{c}}) := \sum_{\substack{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x}) = \mathbf{c}}} w(\theta_Q^\alpha).$$

Clearly, by definition, this extension of w satisfies the weight constraint

$$\nu_Q^{\mathbf{x} \doteq \mathbf{c}} \doteq \sum_{\substack{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x}) = \mathbf{c}}} \theta_Q^\alpha$$

that appears in $\langle repl^\nu(L) \rangle^{\mathbb{D}}$. Moreover, for any sum expression S of L , $eval_w(\langle S \rangle^{\mathbb{D}})$ will give the same value as $eval_w(\langle repl^\nu(S) \rangle^{\mathbb{D}})$ as every weight expression W of S has been replaced in $repl^\nu(S)$ by $\nu(W)$ and that from what precedes $w(\nu(W))$ has the same value as $eval_w(\langle W \rangle^{\mathbb{D}})$. Hence, any solution of $\langle L \rangle^{\mathbb{D}}$ can be extended to a solution of $\langle repl^\nu(L) \rangle^{\mathbb{D}}$ that evaluates to the same objective value.

On the other hand, with the same reasoning, any solution w of $\langle repl^\nu(L) \rangle^{\mathbb{D}}$ directly gives a solution of $\langle L \rangle^{\mathbb{D}}$ that evaluates to the same objective value since the weight constraints ensure that $eval_w(\langle W \rangle^{\mathbb{D}}) = w(\nu(W))$. \square

3.4. Interpretations of linear programs. In this section, we formalize the notion of interpretation of linear programs in $LP_{clos}(CQ_\Sigma)$ and give necessary conditions for an alternate interpretation to be sound. In the following, we denote by $Queries_w(L)$ the set of conjunctive queries that appear in L , that is, the set of Q such that there is an expression of the form $\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q)$ in L .

An *interpretation* $I = (I_W, I_C)$ of $LP_{clos}(CQ_\Sigma)$ is a pair of functions such that, given a database \mathbb{D} over signature Σ :

- I_W maps every weight expression $W := \mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q)$ and database \mathbb{D} to a linear sum $I_W^{\mathbb{D}}(W)$ over linear program variables $X_{I,Q}^{\mathbb{D}}$,
- I_C maps every conjunctive query Q over signature Σ and database \mathbb{D} to a set of constraints $I_C^{\mathbb{D}}(Q)$ on variables $X_{I,Q}^{\mathbb{D}}$,
- and for every two conjunctive queries Q, Q' , if $Q \neq Q'$ then $X_{I,Q}^{\mathbb{D}} \cap X_{I,Q'}^{\mathbb{D}} = \emptyset$.

Given an interpretation I , a conjunctive query Q and a set of weight constraints \mathbf{W} over Q , we denote by:

$$I_Q^{\mathbb{D}}(\mathbf{W}) := I_C^{\mathbb{D}}(Q) \wedge \bigwedge_{W \in \mathbf{W}} \nu(W) = I_W^{\mathbb{D}}(W)$$

where ν is a fixed function as constructed in Section 3.3. Observe that when $Q' \neq Q$, then $I_Q^{\mathbb{D}}(\mathbf{W})$ and $I_{Q'}^{\mathbb{D}}(\mathbf{W}')$ have disjoint variables.

Given a linear program $L = \mathbf{maximize} \ S \ \mathbf{subject\ to} \ C$ with some linear sum $S \in LS_{clos}(CQ_\Sigma)$ and some linear constraint $C \in LC_{clos}(CQ_\Sigma)$, we denote by $\mathbf{W}_Q(L)$ the set of weight expressions of L over Q . The *I-interpretation of L* is the following linear program $I^{\mathbb{D}}(L)$:

$$I^{\mathbb{D}}(L) = \begin{array}{ll} \mathbf{maximize} & \langle S \rangle^\nu \\ \mathbf{subject\ to} & \langle C \rangle^\nu \wedge \\ & \bigwedge_{Q \in Queries_w(L)} I_Q^{\mathbb{D}}(\mathbf{W}_Q(L)). \end{array}$$

For example, the replacement rewriting of a linear program in L can be defined by the interpretation $N = (N_W, N_C)$ such that $N_C^{\mathbb{D}}(Q) := \text{true}$ and

$$N_W(\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q))^{\mathbb{D}} := \sum_{\substack{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x}) = \mathbf{c}}} \theta_Q^\alpha.$$

It is readily verified that $N^{\mathbb{D}}(L)$ corresponds to $\text{repl}^\nu(L)$.

Since, for $Q \neq Q'$, the variables of $I_Q^{\mathbb{D}}(L)$ and of $I_{Q'}^{\mathbb{D}}(L)$ are disjoint, it allows us to prove the following sufficient condition for an interpretation to be sound, that only depends on the value of the interpretation on weight expressions over one conjunctive query Q :

Proposition 3.5. *Let $I = (I_W, I_C)$ be an interpretation of $LP_{\text{clos}}(CQ_\Sigma)$ such that for every conjunctive query Q and set \mathbf{W} of weight expressions over Q we have that $\llbracket I_Q^{\mathbb{D}}(\mathbf{W}) \rrbracket_{|\nu(\mathbf{W})} = \llbracket \langle \text{wc}^\nu(\mathbf{W}) \rangle^{\mathbb{D}} \rrbracket_{|\nu(\mathbf{W})}$. Then I is sound, that is, for every $L \in LP_{\text{clos}}(CQ_\Sigma)$ and database \mathbb{D} , $\text{opt}(\langle L \rangle^{\mathbb{D}}) = \text{opt}(I^{\mathbb{D}}(L))$.*

Proof. Let $L = \mathbf{maximize} \ S \ \mathbf{subject} \ \mathbf{to} \ C$. By Proposition 3.4,

$$\text{opt}(\langle L \rangle^{\mathbb{D}}) = \text{opt}(\langle \text{repl}^\nu(L) \rangle^{\mathbb{D}}).$$

Hence it is sufficient to show that $\text{opt}(\langle \text{repl}^\nu(L) \rangle^{\mathbb{D}}) = \text{opt}(I^{\mathbb{D}}(L))$.

Now recall that $\langle \text{repl}^\nu(L) \rangle^{\mathbb{D}}$ and $I^{\mathbb{D}}(L)$ have the same objective function $\langle S \rangle^\nu$ and the same constraint $\langle C \rangle^\nu$ on variables $\nu(\mathbf{W})$. Only the last part of the program is different.

$I^{\mathbb{D}}(L)$ contains additional constraints $\bigwedge_{Q \in \text{Queries}_w(L)} I_Q^{\mathbb{D}}(\mathbf{W}_Q(L))$ and $\langle \text{repl}^\nu(L) \rangle^{\mathbb{D}}$ contains $\langle \text{wc}^\nu(\mathbf{W}(L)) \rangle^{\mathbb{D}}$.

However, the assumption from the statement applied to every $Q \in \text{Queries}_w(L)$ shows that

$$\llbracket \langle \text{wc}^\nu(\mathbf{W}(L)) \rangle^{\mathbb{D}} \rrbracket_{|\nu(\mathbf{W})} = \llbracket \bigwedge_{Q \in \text{Queries}_w(L)} I_Q^{\mathbb{D}}(\mathbf{W}_Q(L)) \rrbracket_{|\nu(\mathbf{W})}$$

since every $I_Q^{\mathbb{D}}(\mathbf{W}_Q(L))$ contains disjoint variables.

It thus means that for any solution w of constraints $\langle \text{wc}^\nu(\mathbf{W}(L)) \rangle^{\mathbb{D}}$, one can construct a solution w' of constraints $\bigwedge_{Q \in \text{Queries}_w(L)} I_Q^{\mathbb{D}}(\mathbf{W}_Q(L))$ that assign variables $\nu(\mathbf{W})$ to the same values. Hence, we have $\text{eval}_{w'}(\langle S \rangle^\nu) = \text{eval}_w(\langle S \rangle^\nu)$. Moreover, since $\text{eval}_w(\langle C \rangle^\nu)$ is true by definition, and since w and w' coincide on ν variables and that $\langle C \rangle^\nu$ only contains $\nu(\mathbf{W})$ variables, $\text{eval}_{w'}(\langle C \rangle^\nu)$ is also true. Hence w' is a solution of $I^{\mathbb{D}}(L)$ and the values of both linear programs on w and w' respectively coincide. Taking w so that it is optimal for $\langle \text{repl}^\nu(L) \rangle^{\mathbb{D}}$ yields that $\text{opt}(\langle \text{repl}^\nu(L) \rangle^{\mathbb{D}}) \leq \text{opt}(I^{\mathbb{D}}(L))$.

On the other hand, we also have that given a solution w' of constraints

$$\bigwedge_{Q \in \text{Queries}_w(L)} I_Q^{\mathbb{D}}(\mathbf{W}_Q(L)),$$

one can construct a solution w of constraints $\langle \text{wc}^\nu(\mathbf{W}(L)) \rangle^{\mathbb{D}}$ that assign variables $\nu(\mathbf{W})$ to the same value. By the same reasoning, it implies that $\text{opt}(\langle \text{repl}^\nu(L) \rangle^{\mathbb{D}}) \geq \text{opt}(I^{\mathbb{D}}(L))$, and the equality follows. \square

Example. To illustrate the notion of interpretations, we will consider a toy alternative interpretation $I = (I_W, I_C)$ defined as follows: the linear program variables $X_{I,Q}^{\mathbb{D}}$ are defined as $\{x_Q^\alpha \mid \alpha \in \llbracket Q \rrbracket^{\mathbb{D}}\}$ and I_W is defined as:

$$I_W^{\mathbb{D}}(\text{weight}_{\mathbf{x}=\mathbf{c}}(Q)) = \sum_{\substack{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x})=\mathbf{c}}} 3 \times x_Q^\alpha$$

We also define $I_C^{\mathbb{D}}$ as

$$\sum_{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}}} x_Q^\alpha \geq 0$$

The interpretation $I^{\mathbb{D}}(L)$ of the example given in Section 3.1 is thus:

$$\begin{array}{ll} \text{maximize} & \nu_2 \\ \text{subject to} & \nu_0 \leq 1 \wedge \\ & \nu_1 \leq 1 \wedge \end{array}$$

$$\begin{array}{l} \nu_0 \doteq 3x_Q^{[x/0,y/0]} + 3x_Q^{[x/0,y/1]} \wedge \\ \nu_1 \doteq 3x_Q^{[x/1,y/0]} + 3x_Q^{[x/1,y/1]} \wedge \\ \nu_2 \doteq 3x_Q^{[x/0,y/0]} + 3x_Q^{[x/0,y/1]} + 3x_Q^{[x/1,y/0]} + 3x_Q^{[x/1,y/1]} \wedge \\ x_Q^{[x/0,y/0]} + x_Q^{[x/0,y/1]} + x_Q^{[x/1,y/0]} + x_Q^{[x/1,y/1]} \geq 0 \end{array}$$

The last constraint corresponds to $I_C^{\mathbb{D}}(Q)$. It is clear that this program has the same optimal value as the original one because it is obtained by substituting θ_Q^α by $3x_Q^\alpha$ and by adding a constraint that is always true since $x_Q^\alpha \geq 0$. Actually, Proposition 3.5 tells us that for every LPCQ L and database \mathbb{D} , $I^{\mathbb{D}}(L)$ has the same optimal value as $\langle L \rangle^{\mathbb{D}}$. Indeed, given a solution w of $\langle wc^\nu(\mathbf{W}) \rangle^{\mathbb{D}}$, we can transform it into a solution w' of $I_Q^{\mathbb{D}}(\mathbf{W})$ defined as $w'(x_Q^\alpha) = \frac{w(\theta_Q^\alpha)}{3}$ and $w'(\nu(W)) = w(\nu(W))$. It is readily verified that w' respects every constraint of $I_C^{\mathbb{D}}(\mathbf{W})$ and that w' has the same values as w on variables $\nu(\mathbf{W})$. Similarly, a solution w' of $I_Q^{\mathbb{D}}(\mathbf{W})$ can be transformed into a solution w of $\langle wc^\nu(\mathbf{W}) \rangle^{\mathbb{D}}$ by defining w as $w(\theta_Q^\alpha) = 3w'(x_Q^\alpha)$ and $w(\nu(W)) = w'(\nu(W))$. Hence $\llbracket I_Q^{\mathbb{D}}(\mathbf{W}) \rrbracket_{|\nu(\mathbf{W})} = \llbracket \langle wc^\nu(\mathbf{W}) \rangle^{\mathbb{D}} \rrbracket_{|\nu(\mathbf{W})}$ and Proposition 3.5 can be applied.

4. SOLVING $LP_{clos}(CQ_\Sigma)$ LINEAR PROGRAMS EFFICIENTLY

In this section, we propose an algorithm for solving linear programs in $LP_{clos}(CQ_\Sigma)$ that is better than the one given in Theorem 3.3. The proof of Theorem 3.2 suggests that the main source of intractability stem from the complexity of answering conjunctive queries, which is reflected in the upper bound given in Theorem 3.3 where the complexity here depends on the worst-case size of the answer sets of queries. However, for many problems on conjunctive queries such as computing the number of answers, one can get better upper bounds by exploiting the fact that they have small fractional hypertree width. By leveraging the notion of fractional hypertree width from conjunctive queries to linear programs of $LP_{clos}(CQ_\Sigma)$, we are able to lower the complexity from $O(|L|^\ell |\mathbb{D}|^{\ell \text{AGM}(L)})$ given by Theorem 3.3 to $O(|L|^\ell |\mathbb{D}|^{\ell \text{fhtw}(L)})$, where $\text{fhtw}(L)$ denotes the (leveraged notion of)

fractional hypertree width of L , when an optimal tree decomposition of L is provided in the input.

To achieve this, we avoid the expensive step of computing $\langle L \rangle^{\mathbb{D}}$ by exploiting tree decompositions \mathcal{T} of the queries of L to generate a smaller linear program $\rho^{\mathcal{T}, \mathbb{D}}(L)$ having only $O(|\mathbb{D}|^{\text{fhtw}(\mathcal{T})})$ variables and show that the optimal value for $\rho^{\mathcal{T}, \mathbb{D}}(L)$ is the same as the optimal value of $\langle L \rangle^{\mathbb{D}}$.

4.1. Tree decomposition of $LP_{\text{clos}}(CQ_{\Sigma})$. We start by lifting the concept of hypertree decompositions from conjunctive queries to linear programs in $LP_{\text{clos}}(CQ_{\Sigma})$. Given a linear program $L \in LP_{\text{clos}}(CQ_{\Sigma})$, recall that we denote by $\text{Queries}_{\text{w}}(L)$ the set of conjunctive queries that appear in L .

Intuitively, our notion of hypertree decomposition for $LP_{\text{clos}}(CQ_{\Sigma})$ will consist in a collection of hypertree decomposition for every $\exists \mathbf{y}.Q \in \text{Queries}_{\text{w}}(L)$. However, we will need a stronger condition on the decomposition than the usual one:

Definition 4.1. Let $L \in LP_{\text{clos}}(CQ_{\Sigma})$, $\exists \mathbf{y}.Q \in \text{Queries}_{\text{w}}(L)$, with Q the quantifier free part of $\exists \mathbf{y}.Q$ and $\text{weight}_{\mathbf{x} \doteq \mathbf{c}}(\exists \mathbf{y}.Q)$ a weight expression of L .

A tree decomposition $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ of $\exists \mathbf{y}.Q$ compatible with $\text{weight}_{\mathbf{x} \doteq \mathbf{c}}(\exists \mathbf{y}.Q)$ is a tree decomposition of Q such that there exists $u \in \mathcal{V}$ with $\text{set}(\mathbf{x}) = \mathcal{B}(u)$.

T is said to be *compatible with L* if it is a tree decomposition of Q compatible with every $\text{weight}_{\mathbf{x} \doteq \mathbf{c}}(\exists \mathbf{y}.Q)$ of L .

Observe that the requirement that a tree decomposition has to be compatible with the linear program may increase the optimal width of the decomposition. For example, consider the conjunctive query $Q = R(x, y) \wedge S(y, z)$. If a linear program L contains an expression of the form $\text{weight}_{x \doteq 0, z \doteq 0}(Q)$, then every tree decomposition of Q compatible with L has width at least $3/2$ whereas optimal tree decomposition for Q have width 1 (a decomposition of Q is given in Section 2.4).

Definition 4.2. Let L be an $LP_{\text{clos}}(CQ_{\Sigma})$ program. A *tree decomposition of L* is defined to be a collection $\mathcal{T}_L = (T_Q)_{Q \in \text{Queries}_{\text{w}}(L)}$ such that T_Q is a tree decomposition of $Q \in \text{Queries}_{\text{w}}(L)$ that is compatible with L .

The *width of \mathcal{T}_L* is defined to be the maximal width of the decomposition trees in \mathcal{T}_L . The *size of \mathcal{T}_L* is defined to be $|\mathcal{T}_L| = \sum_{(\mathcal{V}, \mathcal{E}, \mathcal{B}) \in \mathcal{T}} |\mathcal{V}|$.

4.2. Factorized Interpretation of quantifier free $LP_{\text{clos}}(CQ_{\Sigma})$. In this section, we present a more succinct way of interpreting weight constraints in linear programs in $LP_{\text{clos}}(CQ_{\Sigma})$, called the factorized interpretation, that exploits a tree decomposition of the queries. In this section, we assume that the linear program only contains quantifier free queries. We will explain in Section 4.3 how one can reduce the case with existentially quantified queries to the case of quantifier free queries.

From now on, we fix a linear language $L \in LP_{\text{clos}}(CQ_{\Sigma})$ and \mathcal{T} a tree decomposition of L . We will describe an interpretation $\rho^{\mathcal{T}} = (\rho_W^{\mathcal{T}}, \rho_C^{\mathcal{T}})$ of L exploiting tree decompositions.

Interpreting weight constraints. Given a conjunctive query $Q \in \text{Queries}_w(L)$ and $T = \mathcal{T}_Q$ the tree decomposition of Q compatible with L given by \mathcal{T} , we define an interpretation of weight expressions on the following set of variables:

$$\Xi_{Q,T}^{\mathbb{D}} := \{\xi_{Q,u}^{\beta} \mid u \in \mathcal{V}, \beta \in \llbracket Q \rrbracket^{\mathbb{D}}|_{\mathcal{B}(u)}\}.$$

Computing the set $\Xi_{Q,T}^{\mathbb{D}}$ can be done efficiently with respect to the width of the decomposition:

Lemma 4.3. *Let k be the width of T . The size of $\Xi_{Q,T}^{\mathbb{D}}$ is at most $|\mathcal{V}| \cdot |\mathbb{D}|^k$ and one can compute $\Xi_{Q,T}^{\mathbb{D}}$ in time $O(|T| \cdot |\mathbb{D}|^k \log(|\mathbb{D}|))$.*

Proof. It follows directly by Lemma 2.4 in Section 2.4. \square

We define the factorized interpretation of the weight expressions $\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q)$ as follows:

$$\rho_W^{T,\mathbb{D}}(\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q)) = \begin{cases} \xi_{Q,u}^{\beta} & \text{if } \beta = [\mathbf{x}/\mathbf{c}] \in \llbracket Q \rrbracket^{\mathbb{D}}|_{\mathcal{B}(u)} \\ 0 & \text{else} \end{cases}$$

where $u \in \mathcal{V}$ is the vertex such that $\text{set}(\mathbf{x}) = \mathcal{B}(u)$ that is the closest to the root of T^3 . If no such u exists then $\rho_W^{T,\mathbb{D}}(\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q))$ is undefined. However, for every weight expression of L , the existence of u is implied by the fact that T is compatible with L , see Definition 4.1.

For $Q \in \text{Queries}_w(L)$, we define $\rho_W^{T,\mathbb{D}}(\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q)) := \rho_W^{\mathcal{T}_Q,\mathbb{D}}(\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q))$. It gives a function to interpret weight expressions in the sense given in Section 3.4 since if Q and Q' are distinct conjunctive queries, then $\rho_W^{T,\mathbb{D}}(\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q))$ and $\rho_W^{T,\mathbb{D}}(\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q'))$ will contain disjoint linear program variables.

Local soundness constraints. If one simply defines the factorized interpretation as $(\rho_W^{\mathcal{T}}, \text{true})$, it will not give a sound interpretation. We illustrate this phenomenon on the example from Section 3.1 using the tree decomposition having three nodes r, u, v rooted at r with r connected to u and v and with $\mathcal{B}(r) = \emptyset$, $\mathcal{B}(u) = \{x\}$ and $\mathcal{B}(v) = \{y\}$. Interpreting only the weights without additional constraints would yield the following:

$$\begin{array}{ll} \mathbf{maximize} & \nu_2 \\ \mathbf{subject\ to} & \nu_0 \leq 1 \wedge \\ & \nu_1 \leq 1 \wedge \\ & \nu_0 \doteq \xi_{Q,u}^{[x/0]} \wedge \\ & \nu_1 \doteq \xi_{Q,u}^{[x/1]} \wedge \\ & \nu_2 \doteq \xi_{Q,r}^{\square}. \end{array}$$

One strange aspect of this program is that it does not depend on the variables $\xi_{Q,v}^{[y/c]}$ and hence on the value of y , because the program does not contain weight expression on variable y . It can be easily checked that the optimal value of this program is not the same as the one from Section 3.1. Indeed, in the above linear program, the values of ν_0, ν_1 and ν_2

³Actually, any vertex u such that $\text{set}(\mathbf{x}_i) = \mathcal{B}(u)$ would work but we choose it to be the closest to the root to have a deterministic definition of the factorized interpretation. It is well-defined by connectedness of tree decompositions. Indeed, if two bags B, B' contain a set S , then their least common ancestor also contains S .

are now completely independent. Hence, the optimal value of the above linear program is actually unbounded.

To make the factorized interpretation equivalent to the natural interpretation, one has to restore somehow the forgotten dependencies. One way of resolving it in the above program would be to add a new constraint $\xi_{Q,r}^\square \doteq \xi_{Q,u}^{[x/0]} + \xi_{Q,u}^{[x/1]}$. To achieve this, we add so-called *local soundness constraints*. For every edge $e = (u, v) \in \mathcal{E}$ of T and $\gamma \in \llbracket Q \rrbracket_{\mathcal{B}(u) \cap \mathcal{B}(v)}^{\mathbb{D}}$, we define the equality constraint $E_\gamma^{e, \mathbb{D}}(Q)$ as follows:

$$\sum_{\substack{\beta \in \llbracket Q \rrbracket_{\mathcal{B}(u)}^{\mathbb{D}} \\ \gamma = \beta|_{\mathcal{B}(u)}}} \xi_{Q,u}^\beta \doteq \sum_{\substack{\beta' \in \llbracket Q \rrbracket_{\mathcal{B}(v)}^{\mathbb{D}} \\ \gamma = \beta'|_{\mathcal{B}(v)}}} \xi_{Q,v}^{\beta'}.$$

Intuitively, this constraint encodes the following: for expressions $\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q)$ and $\mathbf{weight}_{\mathbf{y} \doteq \mathbf{c}'}(Q)$, if the assignments $\beta = [\mathbf{x}/\mathbf{c}]$ and $\beta' = [\mathbf{y}/\mathbf{c}']$ are compatible with one another, in the sense that they agree on the common variables they assign, then $\langle \mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q) \rangle^{\mathbb{D}}$ and $\langle \mathbf{weight}_{\mathbf{y} \doteq \mathbf{c}'}(Q) \rangle^{\mathbb{D}}$ will contain common variables from $\Theta_Q^{\mathbb{D}}$ and hence, will not produce independent linear sums. On the other hand, in the factorized interpretation without additional constraint, they will be interpreted as independent variables $\xi_{Q,u}^\beta$ and $\xi_{Q,v}^{\beta'}$ for some u, v in \mathcal{V} . If $e = (u, v)$ is an edge of T , then $E_\gamma^{e, \mathbb{D}}(Q)$ accounts for the missed dependency in the factorized interpretation. It turns out that these constraints are enough to make the factorized interpretation equivalent to the natural interpretation (in the sense that they have the same optimal value).

Hence, we define $\rho_C^{T, \mathbb{D}}(Q)$, the *local soundness constraints of Q w.r.t T* , as follows:

$$\bigwedge_{e=(u,v) \in \mathcal{E}} \bigwedge_{\gamma \in \llbracket Q \rrbracket_{\mathcal{B}(u) \cap \mathcal{B}(v)}^{\mathbb{D}}} E_\gamma^{e, \mathbb{D}}(Q)$$

Local soundness constraints can be efficiently computed with respect to the width of the decomposition:

Lemma 4.4. *Let k be the width of T . The size of $\rho_C^{T, \mathbb{D}}(Q)$ is at most $|T| \cdot |\mathbb{D}|^k$ and one can compute $\rho_C^{T, \mathbb{D}}(Q)$ in time $O(|Q| \cdot |T| \cdot |\mathbb{D}|^k \log(|\mathbb{D}|))$.*

Proof. There is one constraint $E_\gamma^{e, \mathbb{D}}(Q)$ in $\rho_C^{T, \mathbb{D}}(Q)$ for every edge e of T and $\gamma \in \llbracket Q \rrbracket_{\mathcal{B}(u) \cap \mathcal{B}(v)}^{\mathbb{D}}$. Since $\mathcal{B}(u) \cap \mathcal{B}(v) \subseteq \mathcal{B}(u)$, $\llbracket Q \rrbracket_{\mathcal{B}(u) \cap \mathcal{B}(v)}^{\mathbb{D}}$ is smaller than $\llbracket Q \rrbracket_{\mathcal{B}(u)}^{\mathbb{D}}$ which is itself smaller than $|\mathbb{D}|^k$ by Lemma 2.4. Hence there are at most $|T| \cdot |\mathbb{D}|^k$ constraints in $\rho_C^{T, \mathbb{D}}(Q)$.

Now, to compute $\rho_C^{T, \mathbb{D}}(Q)$, we start by computing $(\llbracket Q \rrbracket_{\mathcal{B}(u)}^{\mathbb{D}})_{u \in \mathcal{V}}$ using Lemma 2.4. Now for each edge $e = (u, v)$ of T , we construct $E_\gamma^{e, \mathbb{D}}(Q)$ as follows: we start by enumerating the tuples $\beta \in \llbracket Q \rrbracket_{\mathcal{B}(u)}^{\mathbb{D}}$ and let $\gamma = \beta|_{\mathcal{B}(v)}$. If it is not yet constructed, we create an empty linear sum S_u^γ and add $\xi_{Q,u}^\beta$ in it. If S_u^γ has already been created, we append $+\xi_{Q,u}^\beta$ to it. We do the same by enumerating $\beta' \in \llbracket Q \rrbracket_{\mathcal{B}(v)}^{\mathbb{D}}$ and let $\gamma = \beta'|_{\mathcal{B}(u)}$. If it is not yet constructed, we create an empty linear sum S_v^γ and add $\xi_{Q,v}^{\beta'}$ in it. If S_v^γ already exists, we just append $+\xi_{Q,v}^{\beta'}$ to it. Finally, we let $E_\gamma^{e, \mathbb{D}}(Q)$ be $S_u^\gamma \doteq S_v^\gamma$ for each $\gamma \in \llbracket Q \rrbracket_{\mathcal{B}(u) \cap \mathcal{B}(v)}^{\mathbb{D}}$ that have been found.

To construct it, observe that each $\llbracket Q \rrbracket_{\mathcal{B}(u)}^{\mathbb{D}}$ is listed at most once for each edges of T and that the projection γ can be constructed in time $O(|Q|)$. Hence, the total time required to construct $\rho_C^{T,\mathbb{D}}(Q)$ is $O(|Q| \cdot |T| \cdot |\mathbb{D}|^k \log(|\mathbb{D}|))$. \square

Factorized interpretation. Taking local soundness constraints into account, we can now define the \mathcal{T} -factorized interpretation $\rho^{\mathcal{T}}$ of a linear program L as the pair $(\rho_W^{\mathcal{T}}, \rho_C^{\mathcal{T}})$ where $\rho_C^{T,\mathbb{D}}(Q) = \rho_C^{\mathcal{T},\mathbb{D}}(Q)$. When \mathcal{T} is clear from context, we will simply say “the factorized interpretation”.

The soundness of factorized interpretation follows from Proposition 3.5 and from results on weighting answer sets of conjunctive queries that can be of independent interest and that we give in Section 6.1:

Theorem 4.5. *Let L be a $LP_{\text{clos}}(CQ_{\Sigma})$ program such that every conjunctive query in $\text{Queries}_w(L)$ is quantifier free, \mathcal{T} a decomposition of L and \mathbb{D} a database. The factorized interpretation $\rho^{\mathcal{T},\mathbb{D}}(L)$ then has the same optimal value as $\langle L \rangle^{\mathbb{D}}$:*

$$\text{opt}(\rho^{\mathcal{T},\mathbb{D}}(L)) = \text{opt}(\langle L \rangle^{\mathbb{D}})$$

Moreover, given an optimal solution W of $\rho^{\mathcal{T},\mathbb{D}}(L)$, there exists a canonical optimal solution ω of $\langle L \rangle^{\mathbb{D}}$ such that given W and a variable θ of $\langle L \rangle^{\mathbb{D}}$, one can compute $\omega(\theta)$ in polynomial time.

The first part of Theorem 4.5 is proven by providing two explicit transformations: one to go from a solution of $\rho^{\mathcal{T},\mathbb{D}}(L)$ to a solution of $\langle L \rangle^{\mathbb{D}}$ having the same value and another one to go from a solution of $\langle L \rangle^{\mathbb{D}}$ to a solution of $\rho^{\mathcal{T},\mathbb{D}}(L)$ having the same value. The preservation of the values by these transformations is enough to establish that both linear programs have the same optimal value. More interestingly, it also allows us to prove the second part of the theorem, that is, that one can recover an optimal solution of $\langle L \rangle^{\mathbb{D}}$ from an optimal solution of $\rho^{\mathcal{T},\mathbb{D}}(L)$. This transformation is described in the proof of Theorem 6.13.

Example. Going back to the example from Section 3.1, and the tree decomposition previously mentioned, $\rho^{\mathcal{T},\mathbb{D}}(L)$ is the following program:

$$\begin{array}{ll} \mathbf{maximize} & \nu_2 \\ \mathbf{subject\ to} & \nu_0 \leq 1 \wedge \\ & \nu_1 \leq 1 \wedge \\ & \nu_0 \doteq \xi_{Q,u}^{[x/0]} \wedge \\ & \nu_1 \doteq \xi_{Q,u}^{[x/1]} \wedge \\ & \nu_2 \doteq \xi_{Q,r}^{\square} \wedge \\ & \xi_{Q,r}^{\square} \doteq \xi_{Q,u}^{[x/0]} + \xi_{Q,u}^{[x/1]} \wedge \\ & \xi_{Q,r}^{\square} \doteq \xi_{Q,v}^{[y/0]} + \xi_{Q,v}^{[y/1]} \end{array}$$

The two last lines contain the local soundness constraint $\rho_C^{T,\mathbb{D}}(Q)$. The last constraint mentions variables $\xi_{Q,v}^{[y/0]}$ and $\xi_{Q,v}^{[y/1]}$ that are not used elsewhere in the program and can safely

be ignored when looking for the optimal value. The other soundness constraint directly implies that $\nu_2 = \nu_0 + \nu_1$ and hence, the optimal value for this program is 2.

Computing the factorized interpretation. The factorized interpretation $\rho^{\mathcal{T}, \mathbb{D}}(L)$ is interesting because it is smaller than the natural interpretation $\langle L \rangle^{\mathbb{D}}$. Indeed, while $\langle L \rangle^{\mathbb{D}}$ has $O(|\mathbb{D}|^{\text{AGM}(L)})$ variables and $O(|L|)$ constraints (see Section 3.2), one can show that if k is the width of \mathcal{T} , then the size of $\rho^{\mathcal{T}, \mathbb{D}}(L)$ is $O(|\mathbb{D}|^k)$ in the data complexity model (where L is considered constant). It follows from the following, more precise, combined complexity analysis:

Theorem 4.6. *Given a relational signature Σ , $L \in LP_{\text{clos}}(CQ_{\Sigma})$ such that every query in $\text{Queries}_{\text{w}}(L)$ is quantifier free, a tree decomposition \mathcal{T} of L and a database \mathbb{D} , we let k be the width of \mathcal{T} , t be the sum of the sizes of the tree decompositions in \mathcal{T} and q be the sum of the sizes of the queries in $\text{Queries}_{\text{w}}(L)$. Then $\rho^{\mathcal{T}, \mathbb{D}}(L)$ has at most $O(t \cdot |\mathbb{D}|^k)$ variables, $O(|L| + t \cdot |\mathbb{D}|^k)$ constraints and can be computed in time $O(|L| + qt \cdot |\mathbb{D}|^k \log |\mathbb{D}|)$. Moreover, $\rho^{\mathcal{T}, \mathbb{D}}(L)$ has size $O(|L||\mathbb{D}|^k)$.*

Proof. This is a direct consequence of applying Lemma 4.3 and Lemma 4.4 to each query in $\text{Queries}_{\text{w}}(L)$. Concerning the size of $\rho^{\mathcal{T}, \mathbb{D}}(L)$, it comes from the fact that each **weight** construct has been replaced by at most one variables resulting in a program of size at most $|L|$ and then we add $O(t|\mathbb{D}|^k) = O(|L||\mathbb{D}|^k)$ soundness constraints. \square

Theorem 4.6 together with Theorem 4.5 implies that the data complexity of computing the optimal value of a linear program $L \in LP_{\text{clos}}(CQ_{\Sigma})$ having only quantifier free queries is below $O(|\mathbb{D}|^{\ell \cdot \text{fhtw}(L)})$ with $\ell < 2.37286$ which improves the complexity stated in Theorem 3.3.

We observe however that the factorized interpretation may be small in practice than the worst-case theoretical bound given by the fractional hypertree width of L . Indeed, the number of variables in the factorized interpretation is the sum of the sizes of $\llbracket Q \rrbracket^{\mathbb{D}}_{\mathcal{B}(u)}$ for each u . In particular, each one of them contains less elements than $\llbracket Q \rrbracket^{\mathbb{D}}$. Hence, even when $\llbracket Q \rrbracket^{\mathbb{D}}$ is small with respect to the worst case, the factorized interpretation will also be smaller than the worst case.

We summarize this discussion in the following theorem, which mirrors Theorem 3.3:

Theorem 4.7. *Given a relational signature Σ , $L \in LP_{\text{clos}}(CQ_{\Sigma})$, a database \mathbb{D} and optimal tree decompositions \mathcal{T} of L of width $\text{fhtw}(L)$, one can compute $\text{opt}(\langle L \rangle^{\mathbb{D}})$ in time $O(|L|^{\ell} |\mathbb{D}|^{\ell \cdot \text{fhtw}(L)})$ with $\ell < 2.37286$.*

4.3. Linear Programs with Existentially Quantified Conjunctive Queries. One drawback of Theorem 4.5 is that it only works for a linear program L in $LP_{\text{clos}}(CQ_{\Sigma})$ containing only quantifier free conjunctive queries. This restriction can actually be lifted since replacing existentially quantified queries of L with their quantifier free part yield a linear program that has the same optimal value under the natural interpretation for any database \mathbb{D} . We formalize this approach and prove its correctness in this section.

Given a quantified conjunctive query $Q' = \exists \mathbf{y}.Q$ with Q a quantifier free conjunctive query, we denote by $gf(Q')$ the conjunctive query $Q \wedge \mathbf{y} \doteq \mathbf{y}$. Clearly, for any database \mathbb{D} , the answer set $\llbracket Q \rrbracket^{\mathbb{D}}$ is the same as $\llbracket gf(Q') \rrbracket^{\mathbb{D}}$ and $gf(Q')$ and Q have the same hypertree decompositions and hence the same width. However, in the following, we will need to be able to syntactically distinguish two queries having the same quantifier free part but different

quantifier prefix and $qf(Q')$ allows us to do so. Indeed, $qf(Q')$ is syntactically different from $Q' = \exists \mathbf{y}'. Q$ for different \mathbf{y} and \mathbf{y}' .

We lift the definition of $qf(Q)$ to a linear program in $L \in LP_{clos}(CQ_\Sigma)$: we denote by $qf(L)$ the $LP_{clos}(CQ_\Sigma)$ language obtained by replacing every expression $\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q')$ in L by $\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(qf(Q'))$. It is clear that $qf(L)$ now only contains quantifier free conjunctive query.

Example. We adapt the example from Section 3.1 by considering the conjunctive query $Q' = \exists y. R_1(x) \wedge R_2(y)$ and the following program $L' \in LP_{clos}(CQ_\Sigma)$:

$$\begin{aligned} & \mathbf{maximize} && \mathbf{weight}_\emptyset(Q') \\ & \mathbf{subject\ to} && \mathbf{weight}_{x \doteq 0}(Q') \leq 1 \\ & && \wedge \mathbf{weight}_{x \doteq 1}(Q') \leq 1 \end{aligned}$$

Over the database given in Section 3.1, we have $\langle L' \rangle^{\mathbb{D}}$:

$$\begin{aligned} & \mathbf{maximize} && \theta_{Q'}^0 + \theta_{Q'}^1 \\ & \mathbf{subject\ to} && \theta_{Q'}^0 \leq 1 \\ & && \wedge \theta_{Q'}^1 \leq 1 \end{aligned}$$

It has optimal value 2.

On the other hand, we have $qf(Q') = R_1(x) \wedge R_2(y) \wedge y \doteq y$ and $qf(L')$ is:

$$\begin{aligned} & \mathbf{maximize} && \mathbf{weight}_\emptyset(qf(Q')) \\ & \mathbf{subject\ to} && \mathbf{weight}_{x \doteq 0}(qf(Q')) \leq 1 \\ & && \wedge \mathbf{weight}_{x \doteq 1}(qf(Q')) \leq 1 \end{aligned}$$

which is clearly equivalent from the linear program L from Section 3.1 which itself has optimal value 2 when interpreted on the database given in the same section.

Soundness of quantifier elimination. It turns out that it is equivalent to L in the following sense:

Proposition 4.8. *For every $L \in LP_{clos}(CQ_\Sigma)$ and \mathbb{D} on signature Σ , we have*

$$\mathit{opt}(\langle qf(L) \rangle^{\mathbb{D}}) = \mathit{opt}(\langle L \rangle^{\mathbb{D}}).$$

Proof. By Proposition 3.4, it is sufficient to show

$$\mathit{opt}(\langle \mathit{repl}^\nu(qf(L)) \rangle^{\mathbb{D}}) = \mathit{opt}(\langle \mathit{repl}^\nu(L) \rangle^{\mathbb{D}}).$$

We start by observing that $\mathbf{W}_Q(L)$ is in one to one correspondence with $\mathbf{W}_{qf(Q)}(qf(L))$ by definition since we replaced every occurrence of Q in L by $qf(Q)$ in $qf(L)$. Moreover, observe that if Q and Q' are distinct queries of $\mathbf{Queries}_w(L)$, then $qf(Q)$ and $qf(Q')$ are also distinct. Indeed, either Q and Q' have distinct quantifier free part and it is obvious, or they have distinct quantifier prefix \mathbf{y} and \mathbf{y}' respectively, in which case $qf(Q)$ and $qf(Q')$ will be distinct since $qf(Q)$ contains $\mathbf{y} \doteq \mathbf{y}$ and $qf(Q')$ contains $\mathbf{y}' \doteq \mathbf{y}'$.

Hence, exploiting this one to one correspondence, for

$$L = \mathbf{maximize\ } S \mathbf{\ subject\ to\ } C$$

with some linear sum $S \in LS_{clos}(CQ_\Sigma)$ and some linear constraint $C \in LC_{clos}(CQ_\Sigma)$, we can see $\langle \mathit{repl}^\nu(qf(L)) \rangle^{\mathbb{D}}$ as:

$$\text{maximize } \langle S \rangle^\nu \text{ subject to } \langle C \rangle^\nu \wedge \bigwedge_{W \in \mathbf{W}(L)} \nu(W) = M^{\mathbb{D}}(W)$$

where $M^{\mathbb{D}}$ maps weight expressions of L to a sum expression. In other words, one can see that $\langle repl^\nu(qf(L)) \rangle^{\mathbb{D}}$ as $I^{\mathbb{D}}(L)$ where $I = (M, true)$ is the alternate interpretation in the sense given in Section 3.4 defined as follows: for a weight expression $W = \mathbf{weight}_{\mathbf{x}=\mathbf{c}}(Q')$ where $Q' = \exists \mathbf{y}.Q$ and Q is quantifier free, $M^{\mathbb{D}}(W)$ is defined over variables

$$X_{Q'}^{\mathbb{D}} := \{\chi_{Q'}^\alpha \mid \alpha \in \llbracket Q \rrbracket^{\mathbb{D}}\}$$

by the following sum expression:

$$M_{Q'}^{\mathbb{D}}(W) := \sum_{\substack{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x})=\mathbf{c}}} \chi_{Q'}^\alpha.$$

By Proposition 3.5, it is thus sufficient to show that for a set \mathbf{W} of weight expressions over a conjunctive query $Q' = \exists \mathbf{y}.Q$ with Q a quantifier free conjunctive query, we have $\llbracket I_{Q'}^{\mathbb{D}}(\mathbf{W}) \rrbracket_{|\nu} = \llbracket \langle wc^\nu(\mathbf{W}) \rangle^{\mathbb{D}} \rrbracket_{|\nu}$. We let V be the free variables of Q' .

So let $w \in \llbracket I_{Q'}^{\mathbb{D}}(\mathbf{W}) \rrbracket$. It maps variables $\chi_{Q'}^\alpha$ to a positive real number, where α is in $\llbracket Q \rrbracket^{\mathbb{D}}$. For $\beta \in \llbracket Q' \rrbracket^{\mathbb{D}}$, we define $w'(\theta_{Q'}^\beta) := \sum_{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \mid \alpha|_V = \beta} w(\chi_{Q'}^\alpha)$. For a weight constraint W ,

we set $w'(\nu(W)) = w(\nu(W))$. Clearly, w and w' coincide on ν variables. It remains to show that $w' \in \llbracket \langle wc^\nu(\mathbf{W}) \rangle^{\mathbb{D}} \rrbracket$, that is, for a weight expression $W \in \mathbf{W}$ of the form $\mathbf{weight}_{\mathbf{x}=\mathbf{c}}(Q')$, $w'(\nu(W)) = w'(\langle W \rangle^{\mathbb{D}})$, that is:

$$w'(\nu(W)) = \sum_{\substack{\beta \in \llbracket Q' \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x})=\mathbf{c}}} w'(\theta_{Q'}^\beta).$$

By definition of w' , the right-hand side of this equation rewrites to:

$$\begin{aligned} \sum_{\substack{\beta \in \llbracket Q' \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x})=\mathbf{c}}} w'(\theta_{Q'}^\beta) &= \sum_{\substack{\beta \in \llbracket Q' \rrbracket^{\mathbb{D}} \\ \beta(\mathbf{x})=\mathbf{c}}} \sum_{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha|_V = \beta} w(\chi_{Q'}^\alpha) \\ &= \sum_{\substack{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x})=\mathbf{c}}} w(\chi_{Q'}^\alpha) \text{ since } \text{set}(\mathbf{x}) \subseteq V \\ &= w(W) \text{ since } w \in \llbracket I_{Q'}^{\mathbb{D}}(\mathbf{W}) \rrbracket. \end{aligned}$$

Hence, $w' \in \llbracket \langle wc^\nu(\mathbf{W}) \rangle^{\mathbb{D}} \rrbracket$.

For the other way around, let $w' \in \llbracket \langle wc^\nu(\mathbf{W}) \rangle^{\mathbb{D}} \rrbracket$. For $\alpha \in \llbracket Q \rrbracket^{\mathbb{D}}$, we let $\beta = \alpha|_V$ and define $w(\chi_{Q'}^\alpha) := \frac{1}{N_\beta} w'(\theta_{Q'}^\beta)$, where N_β is the number of $\gamma \in \llbracket Q \rrbracket^{\mathbb{D}}$ such that $\gamma|_V = \beta$. For every weight expression $W \in \mathbf{W}$, we let $w(\nu(W)) := w'(\nu(W))$. Clearly w and w' coincide on ν variables. Now, it remains to show that $w \in \llbracket I_{Q'}^{\mathbb{D}}(\mathbf{W}) \rrbracket$ that is for a weight expression

$W \in \mathbf{W}$ of the form $\mathbf{weight}_{\mathbf{x}=\mathbf{c}}(Q')$, $w(\nu(W)) = w(M_Q^{\mathbb{D}}(W))$. By definition, we have:

$$\begin{aligned}
w(M_Q^{\mathbb{D}}(W)) &= \sum_{\substack{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x})=\mathbf{c}}} w(\chi_{Q'}^{\alpha}) \\
&= \sum_{\substack{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x})=\mathbf{c}}} \frac{1}{N_{\beta}} w'(\theta_{Q'}^{\beta}) \text{ with } \beta = \alpha|_V \\
&= \sum_{\substack{\beta \in \llbracket Q' \rrbracket^{\mathbb{D}} \\ \beta(\mathbf{x})=\mathbf{c}}} \sum_{\substack{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha|_V=\beta}} \frac{1}{N_{\beta}} w'(\theta_{Q'}^{\beta}) \\
&= \sum_{\substack{\beta \in \llbracket Q' \rrbracket^{\mathbb{D}} \\ \beta(\mathbf{x})=\mathbf{c}}} w'(\theta_{Q'}^{\beta}) \text{ by definition of } N_{\beta} \\
&= w'(\nu(W)) \text{ since } w' \in \llbracket \langle wc^{\nu}(\mathbf{W}) \rangle^{\mathbb{D}} \rrbracket \\
&= w(\nu(W)) \text{ by definition of } w.
\end{aligned}$$

□

Recall that by definition, a hypertree decomposition \mathcal{T} of width k of a linear program L in $LP_{clos}(CQ_{\Sigma})$ consists of a collection of tree decompositions for the quantifier free part of each query in $\text{Queries}_{\mathbf{w}}(L)$ of width at most k . For $Q \in \text{Queries}_{\mathbf{w}}(L)$, \mathcal{T}_Q is then also a decomposition of $qf(Q)$ with width at most k . Hence, \mathcal{T} is a hypertree decomposition of $qf(L)$ of width k . The fractional hypertree width of $qf(L)$ is thus the same as the fractional hypertree width of L . Hence, one can compute the optimal value of L in $O(|\mathbb{D}|^{\ell \cdot \text{fhtw}(L)})$ with $\ell < 2.37286$ in data complexity by computing the optimal value of the factorized interpretation of $qf(L)$ using Theorem 4.5 and Theorem 4.6, even when the program contains existentially quantified conjunctive queries.

This is wrapped up in the following theorem which is an improvement over Theorem 3.3:

Theorem 4.9. *Given a relational signature Σ , $L \in LP_{clos}(CQ_{\Sigma})$, a tree decomposition \mathcal{T} of L of width k and a database \mathbb{D} , there exists some $\ell < 2.37286$ such that one can compute $\text{opt}(\langle L \rangle^{\mathbb{D}})$ in time $O((|L| + tq|\mathbb{D}|^k)^{\ell})$ where t is the sum of the sizes of tree decompositions in \mathcal{T} and q the sum of the sizes of the conjunctive queries in $\text{Queries}_{\mathbf{w}}(L)$.*

5. LINEAR PROGRAMS WITH OPEN WEIGHT EXPRESSIONS

While we have shown that $LP_{clos}(CQ_{\Sigma})$ programs can be solved efficiently by exploiting tree decompositions of the input conjunctive queries, it is not yet powerful enough to express interesting linear program such as the one presented in Section 1.1. The missing feature in $LP_{clos}(CQ_{\Sigma})$ for is their inability to quantify over values in the database to create new constraints. This is especially useful in the example of Section 1.1. The last quantified constraint states that the storing limit of every warehouse in the database will not be exceeded in the solution of the linear program. This expressivity is enabled by the fact that one can universally quantify over warehouses given the table *store*, which will be interpreted

as generating one constraint for each. Moreover, observe that this constraint also draws a numerical value $\mathbf{num}(l)$ from the database.

In this section, we introduce the language $LP(CQ_\Sigma)$ allowing to universally quantify and sum over answers set of database queries. The syntax of the language is presented in Figure 7. It includes definitions for linear sums, linear constraints and linear programs. The semantics of programs in $LP(CQ_\Sigma)$ will be given in Section 5.1 via a closure operation, transforming an $LP(CQ_\Sigma)$ into a program in $LP_{clos}(CQ_\Sigma)$.

| | | | |
|--------------------|---------------------------|-------|--|
| Domain expressions | $E \in Exp(CQ_\Sigma)$ | $::=$ | $x \mid c$ |
| Constant numbers | $N \in Num(CQ_\Sigma)$ | $::=$ | $r \mid \mathbf{num}(E)$ |
| Linear sums | $S, S' \in LS(CQ_\Sigma)$ | $::=$ | $\mathbf{weight}_{\mathbf{z}: \mathbf{x} \doteq \mathbf{y}}(Q)$ where $set(\mathbf{x}) \subseteq fv(Q) \subseteq set(\mathbf{z})$, $set(\mathbf{y})$ contains variables and constants and $set(\mathbf{y}) \cap set(\mathbf{z}) = \emptyset$ |
| | | | $\mid \sum_{\mathbf{x}:Q} S$ |
| | | | $\mid NS \mid S + S' \mid N$ |
| Linear constraints | $C, C' \in LC(CQ_\Sigma)$ | $::=$ | $S \leq S' \mid S \doteq S' \mid C \wedge C' \mid true$ $\mid \forall \mathbf{x}:Q.C$ |
| Linear programs | $L \in LP(CQ_\Sigma)$ | $::=$ | $\mathbf{maximize } S \text{ subject to } C$ where $fv(S) = fv(C) = \emptyset$. |

Figure 7: Linear sum, constraints, and programs with open weight expressions over conjunctive queries $Q, Q' \in CQ_\Sigma$, with variables $x \in X$, sequences of variables $\mathbf{x} \in X^*$, constants $c \in C$, and reals $r \in \mathbb{R}$.

Apart from the addition of an operator $\mathbf{num}(E)$ which will intuitively allow to get numerical constants from the database, universal quantifiers and sums ranging over a conjunctive query, the main difference with $LP_{clos}(CQ_\Sigma)$ programs is that non-constant values are allowed in $\mathbf{weight}_{\mathbf{z}: \mathbf{x} \doteq \mathbf{y}}(Q)$ expressions. We will call such weight expression *open weight expressions*, as opposed to closed weight expressions where \mathbf{y} only contains constants. Intuitively, variables in $set(\mathbf{y})$ will be replaced by database constants in the closure of an $LP(CQ_\Sigma)$.

A valid $LP(CQ_\Sigma)$ program L does not have free variables, that is, every variable has to be bound by one of the new operator: either a linear sum $\sum_{\mathbf{x}:Q} S$ or a universal quantifiers $\forall \mathbf{x}:Q.C$. To formalize this notion, we give in Figure 8 the definition of the free variables of an $LP(CQ_\Sigma)$ program.

Observe in particular that for $\mathbf{weight}_{\mathbf{z}: \mathbf{x} \doteq \mathbf{y}}(Q)$, only variables in $set(\mathbf{y})$ are considered free since $fv(Q) \subseteq set(\mathbf{z})$. The free variables of the conjunctive queries (and hence in $set(\mathbf{x})$ since $set(\mathbf{x}) \subseteq fv(Q)$) are *not* considered free variables. In other words, variables bounded through universal quantifiers and sums over conjunctive queries in $LP(CQ_\Sigma)$ will not introduce constants in conjunctive queries appearing in weight expressions.⁴ If one follows Barendregt's variable convention, it means that the variables appearing in the conjunctive queries of $L \in LP(CQ_\Sigma)$ may be considered disjoint from the variables used in the linear program part.

⁴This restriction was not present in the conference version of this paper [CCNR22] which may lead to counter intuitive behaviors.

$$\begin{array}{ll}
fv(c) = \emptyset & fv(\mathbf{num}(E)) = fv(E) \\
fv(\mathbf{weight}_{\mathbf{z}:\mathbf{x} \doteq \mathbf{y}}(Q)) = (fv(Q) \cup set(\mathbf{y})) \setminus set(\mathbf{z}) & fv(\sum_{\mathbf{x}:Q} S) = fv(S) \cup fv(Q) \setminus set(\mathbf{x}) \\
fv(NS) = fv(N) \cup fv(S) & fv(S \leq S') = fv(S) \cup fv(S') \\
fv(S + S') = fv(S) \cup fv(S') & fv(S \doteq S') = fv(S) \cup fv(S') \\
fv(\forall \mathbf{x}:Q. C) = fv(Q) \cup fv(C) \setminus \{\mathbf{x}\} & fv(C \wedge C') = fv(C) \cup fv(C') \\
fv(\mathbf{maximize} S \mathbf{subject to} C) = \emptyset & fv(true) = \emptyset
\end{array}$$

Figure 8: Free variables of expressions, constraints, and programs, where $var(\mathbf{y})$ denotes the elements of $set(\mathbf{y})$ that are not constants.

5.1. Closure and semantics. We define the semantics of linear programs with open weight expressions over a database \mathbb{D} by mapping it to a linear program with closed weight expression. Intuitively, the queries in $\sum_{\mathbf{x}:Q} S$ and a universal quantifiers $\forall \mathbf{x}:Q.C$ get interpreted over \mathbb{D} and it maps \mathbf{x} to some possible values that passed into a context. The details are given in Figure 9.

$$\begin{array}{l}
close(r)^{\mathbb{D},\gamma} = r \\
close(\mathbf{num}(d))^{\mathbb{D},\gamma} = d \\
close(\mathbf{num}(x))^{\mathbb{D},\gamma} = \mathbf{num}^{\mathbb{D}}(\gamma(x)) \\
close(\mathbf{weight}_{\mathbf{x}:\mathbf{z} \doteq \mathbf{y}}(Q))^{\mathbb{D},\gamma} = \mathbf{weight}_{\mathbf{z} \doteq \tilde{\gamma}(\mathbf{y})}(Q) \\
close(S_1 + S_2)^{\mathbb{D},\gamma} = close(S_1)^{\mathbb{D},\gamma} + close(S_2)^{\mathbb{D},\gamma} \\
close(NS)^{\mathbb{D},\gamma} = close(N)^{\mathbb{D},\gamma} close(S)^{\mathbb{D},\gamma} \\
close(\sum_{\mathbf{x}:Q} S)^{\mathbb{D},\gamma} = \sum_{\gamma' \in \llbracket ext_{\mathbf{x}}(subs_{\tilde{\gamma}}(Q)) \rrbracket^{\mathbb{D}}} close(S)^{\mathbb{D},\tilde{\gamma} \cup \gamma'} \\
close(\forall \mathbf{x}:Q. C)^{\mathbb{D},\gamma} = \bigwedge_{\gamma' \in \llbracket ext_{\mathbf{x}}(subs_{\tilde{\gamma}}(Q)) \rrbracket^{\mathbb{D}}} close(C)^{\mathbb{D},\tilde{\gamma} \cup \gamma'} \\
close(S_1 \leq S_2)^{\mathbb{D},\gamma} = close(S_1)^{\mathbb{D},\gamma} \leq close(S_2)^{\mathbb{D},\gamma} \\
close(C_1 \wedge C_2)^{\mathbb{D},\gamma} = close(C_1)^{\mathbb{D},\gamma} \wedge close(C_2)^{\mathbb{D},\gamma} \\
close(true)^{\mathbb{D},\gamma} = true \\
close(\mathbf{maximize} S \mathbf{subject to} C)^{\mathbb{D}} \\
= \mathbf{maximize} close(S)^{\mathbb{D},\emptyset} \mathbf{subject to} close(C)^{\mathbb{D},\emptyset}
\end{array}$$

Figure 9: Closure $close(F)^{\mathbb{D},\gamma}$ of linear programs F with relational descriptors $L \in LP(CQ_{\Sigma})$ to linear programs with set descriptors $close(L)^{\mathbb{D}} \in LP_{clos}(CQ_{\Sigma})$. Furthermore, $\gamma : Y \rightarrow D$ a variable assignment with $fv(F) \subseteq Y \subseteq \mathcal{X}$, and $\tilde{\gamma} = \gamma|_{Y \setminus set(\mathbf{x})}$. Recall that $subs$ is the substitution operator and $ext_{\mathbf{x}}$ is the extension operator as defined in Section 2.2

The closure is defined in a somewhat classical way: one inductively evaluates a linear program L over a database \mathbb{D} and an environment γ mapping free variables of L to values in the domain. We illustrate this on by detailing the closure of $(\forall \mathbf{x}:Q. C)$ over a database \mathbb{D} given an environment γ . The closure of $\sum_{\mathbf{x}:Q} S$ and of $\mathbf{weight}_{\mathbf{x}:\mathbf{z} \doteq \mathbf{y}}(Q)$ being very similar.

The closure of $(\forall \mathbf{x}:Q. C)$ generates one set of constraint for each answer of Q over \mathbb{D} (over variables \mathbf{x}). It intuitively means that for every answer of Q , we want the constraints of the closure of C to be satisfied. However, when evaluated the closure inductively, Q may contain free variables. These variables must have a value set by γ . We start by mapping the

variables of Q that are free to their value in the environment using $subs_{\tilde{\gamma}}(Q)$, where $\tilde{\gamma}$ is the restriction of γ to the free variables of Q . Indeed, it may be that γ assigns a variable x that appears in \mathbf{x} . In this case, we consider that x is not bounded by γ since it is not free in $(\mathbf{x} : Q)$. For example, $\forall x : Q_1. \forall x : Q_2. C$ has the same closure as $\forall x : Q_1. \forall y : Q_2. C$ since the value of γ over x when computing the closure of $\forall x : Q_2. C$ is not used in $\tilde{\gamma}$. To summarize, the closure of $(\forall \mathbf{x} : Q. C)$ over a database \mathbb{D} in an environment γ generates a set of constraints $close(C)^{\mathbb{D}, \tilde{\gamma} \cup \gamma'}$ for every γ' that is a solution of Q over variables \mathbf{x} where each variable of Q that are not in \mathbf{x} have been replaced by their value under γ , which is formally written as $\gamma' \in \llbracket ext_{\mathbf{x}}(subs_{\tilde{\gamma}}(Q)) \rrbracket^{\mathbb{D}}$.

Observe that the closure is always well-defined due to the syntactic restrictions on linear programs. Also observe that one needs to be able to interpret some database constants as numerical values because of the **num** operation. Hence, one needs to use databases that can contain real numbers. This is formalized in the following definition: a (*relational*) *database with real numbers* is a tuple $\mathbb{D} = (\Sigma, dom(\mathbb{D}), \cdot^{\mathbb{D}}, \mathbf{num}^{\mathbb{D}})$ such that $\mathbb{D} = (\Sigma, dom(\mathbb{D}), \cdot^{\mathbb{D}})$ is a relational Σ -structure and $\mathbf{num}^{\mathbb{D}}$ a partial function from $dom(\mathbb{D})$ to \mathbb{R} .

Since a linear programs $L \in LP(CQ_{\Sigma})$ do not have free variables, the closure of L indeed produces a linear program in $LP_{clos}(CQ_{\Sigma})$, as stated below:

Proposition 5.1. *For any linear programs $L \in LP(CQ_{\Sigma})$ with open weight expressions and database with numerical values $\mathbb{D} = (\Sigma, D, \llbracket \cdot \rrbracket^{\mathbb{D}}, \mathbf{num}^{\mathbb{D}})$ such that $D \subseteq \mathcal{C}$, the closure $close(L)^{\mathbb{D}}$ is a linear program in $LP_{clos}(CQ_{\Sigma})$.*

The *natural interpretation* $\langle L \rangle^{\mathbb{D}}$ of an $LP(CQ_{\Sigma})$ L over a database \mathbb{D} is defined to be the natural interpretation of its closure, that is, $\langle L \rangle^{\mathbb{D}}$ is defined to be $\langle close(L)^{\mathbb{D}} \rangle^{\mathbb{D}}$.

Example. As an example, we reconsider the conjunctive query $Q = R_1(x) \wedge R_2(y)$ from Section 3.1. Assume we also have a unary relation S and a binary relation T in \mathbb{D} with $S^{\mathbb{D}} = \{0, 1\}$ and $T^{\mathbb{D}} = \{(0, 0.4), (0, 0.6), (1, 0.3)\}$. We then consider the following $LP(CQ_{\Sigma})$ program L :

$$\begin{array}{ll} \mathbf{maximize} & \mathbf{weight}_{(x,y):true}(Q) \\ \mathbf{subject\ to} & \forall (z):S(z). \mathbf{weight}_{(x,y):x=z}(Q) \leq \sum_{y:T(z,y)} \mathbf{num}(y) \end{array}$$

To compute the closure over \mathbb{D} of this program, we start by unfolding the universal quantifier. Hence $close(L)^{\mathbb{D}, \llbracket \cdot \rrbracket^{\mathbb{D}}}$ is equal to:

$$\begin{array}{ll} \mathbf{maximize} & \mathbf{weight}_{(x,y):true}(Q) \\ \mathbf{subject\ to} & close(\mathbf{weight}_{(x,y):x=z}(Q))^{\mathbb{D}, z \mapsto 0} \leq close(\sum_{y:T(0,y)} \mathbf{num}(y))^{\mathbb{D}, z \mapsto 0} \\ \wedge & close(\mathbf{weight}_{(x,y):x=z}(Q))^{\mathbb{D}, z \mapsto 1} \leq close(\sum_{y:T(0,y)} \mathbf{num}(y))^{\mathbb{D}, z \mapsto 1} \end{array}$$

By evaluating the closure of the weights and sum we then have:

$$\begin{array}{ll} \mathbf{maximize} & \mathbf{weight}_{(x,y):true}(Q) \\ \mathbf{subject\ to} & \mathbf{weight}_{x=0}(Q) \leq close(\mathbf{num}(y))^{\mathbb{D}, z \mapsto 0, y \mapsto 0.4} + close(\mathbf{num}(y))^{\mathbb{D}, z \mapsto 0, y \mapsto 0.6} \\ \wedge & \mathbf{weight}_{x=1}(Q) \leq close(\mathbf{num}(y))^{\mathbb{D}, z \mapsto 1, y \mapsto 0.3} \end{array}$$

And finally:

$$\begin{array}{ll} \mathbf{maximize} & \mathbf{weight}_{(x,y):true}(Q) \\ \mathbf{subject\ to} & \mathbf{weight}_{x=0}(Q) \leq 0.4 + 0.6 \\ \wedge & \mathbf{weight}_{x=1}(Q) \leq 0.3 \end{array}$$

5.2. Complexity of solving $LP(CQ_\Sigma)$ programs. In this section, we are interested in the complexity of solving $LP(CQ_\Sigma)$ programs.

Hardness. Universal quantifiers make the complexity of solving programs in $LP(CQ_\Sigma)$ much harder than for $LP_{clos}(CQ_\Sigma)$:

Theorem 5.2. *The problem of computing $\langle L \rangle^\mathbb{D}$ for an $LP(CQ_\Sigma)$ L and a database \mathbb{D} with real values given in the input is $\#P$ -hard.*

Proof. Given a conjunctive query Q on variables \mathbf{x} , we define L_Q to be the following $LP(CQ_\Sigma)$ program:

$$\begin{array}{ll} \text{maximize} & \text{weight}_{true}(Q) \\ \text{subject to} & \forall \mathbf{y}: Q(\mathbf{y}). \text{weight}_{\mathbf{x}:\mathbf{x}=\mathbf{y}}(Q) \leq 1. \end{array}$$

We claim that given a database \mathbb{D} , the optimal value of $\langle L \rangle^\mathbb{D}$ is the size of $\llbracket Q \rrbracket^\mathbb{D}$. Indeed, $close(L)^\mathbb{D}$ contains one constraint $\text{weight}_{\mathbf{x}:\mathbf{x}=\alpha(\mathbf{x})}(Q) \leq 1$ for each $\alpha \in \llbracket Q \rrbracket^\mathbb{D}$. Hence, it is translated as a constraint $\theta_Q^\alpha \leq 1$ in $\langle L \rangle^\mathbb{D}$, while the objective function is $\sum_{\alpha \in \llbracket Q \rrbracket^\mathbb{D}} \theta_Q^\alpha$. Assigning every θ_Q^α yields a solution of $\langle L \rangle^\mathbb{D}$ whose value is the size of $\llbracket Q \rrbracket^\mathbb{D}$. Moreover, since every constraint in $\langle L \rangle^\mathbb{D}$ is saturated, it is also optimal.

The problem of computing the size of $\llbracket Q \rrbracket^\mathbb{D}$ when both Q and \mathbb{D} are given in the input is $\#P$ -complete [PS13], hence, computing $opt(\langle L \rangle^\mathbb{D})$ is $\#P$ -hard. \square

Data complexity. When considering the input linear program size to be constant, that is, in the data complexity model, it turns out that one can compute the optimal value of the natural interpretation of an $LP(CQ_\Sigma)$ program L over \mathbb{D} in time polynomial in the size of the database \mathbb{D} . To analyze the precise complexity of solving programs in $LP(CQ_\Sigma)$, we start by defining a normal form for $LP(CQ_\Sigma)$ programs that will be helpful.

An *atomic linear constraint* is a linear constraint of the form $\forall \mathbf{x}: Q. S \leq S'$, $\forall \mathbf{x}: Q. S = S'$, $S \leq S'$ or $S = S'$ where S and S' are linear sums. We insist on the fact that an atomic linear constraint has at most one conjunctive query on which the universal quantifier applies. An *atomic linear sum* is a linear sum of the form N , $N \sum_{\mathbf{z}: Q'} 1$, $\sum_{\mathbf{z}: Q'} \text{weight}_{\mathbf{z}:\mathbf{x}=\mathbf{y}}(Q)$ or $N \sum_{\mathbf{z}: Q'} \text{weight}_{\mathbf{z}:\mathbf{x}=\mathbf{y}}(Q)$ with N a constant number (of the form $r \in \mathbb{R}$ or $\text{num}(E)$).

A linear sum is said to be in *normal form* if it is written as a sum of atomic linear sum. A linear constraint is in normal form if it is written as a conjunction of atomic linear constraints on linear sum in normal form. Finally, an $LP(CQ_\Sigma)$ is in normal form if its objective is a linear sum in normal form and if its constraint is in normal form.

It turns out that every $LP(CQ_\Sigma)$ program can be written as an equivalent linear program in normal form of polynomial size.

Theorem 5.3. *Let L be an $LP(CQ_\Sigma)$. There exists a normal form $LP(CQ_\Sigma)$ L' such that L' is of size at most $|L|^3$ and such that for every database \mathbb{D} , $close(L)^\mathbb{D}$ and $close(L')^\mathbb{D}$ have the same constraints (up to permutations).*

Proof. We first assume that L is written using Barendregt's variable convention [Bar12] to avoid variables capture. First we observe that a linear constraint can always be written as a conjunction of constraints of the form

$$\forall \mathbf{x}_1 : Q_1 \dots \forall \mathbf{x}_k : Q_k. B$$

where B is of the form $S = S'$ or $S \leq S'$ (we consider that $k = 0$ corresponds to the case without quantifier). It comes from the fact that the closure of $\forall \mathbf{x} : Q.(C_1 \wedge C_2)$ will generate the same constraints as $(\forall \mathbf{x} : Q.C_1) \wedge (\forall \mathbf{x} : Q.C_2)$. One can hence apply this transformation until all constraints are of the desired form. The transformation applied to L results in an $LP(CQ_\Sigma)$ L_1 of size at most $d_\forall |L|$ where d_\forall is the depth of universal quantifiers of L , that is, the maximal number of universal quantifiers that are enclosed in one another. Indeed, one can see that L_1 will contain a conjunction of constraints of the form $\forall \mathbf{x}_1 : Q_1 \dots \forall \mathbf{x}_k : Q_k.B$ where B is of the form $S = S'$ or $S \leq S'$ and appears in L enclosed in several quantifiers $\forall \mathbf{x}_1 : Q_1, \dots, \forall \mathbf{x}_k : Q_k$.

Similarly, one can rewrite each linear sum S as a sum of expressions of the form

$$\sum_{\mathbf{x}_1 : Q_1} \dots \sum_{\mathbf{x}_k : Q_k} B$$

where B is of the form N , $\mathbf{weight}_{\mathbf{z}:\mathbf{x}=\mathbf{y}}(Q)$ or $N\mathbf{weight}_{\mathbf{z}:\mathbf{x}=\mathbf{y}}(Q)$ (again, the case $k = 0$ corresponds to the case without \sum). Indeed, we proceed similarly by observing that the closure of $(\sum_{\mathbf{x}:Q}(S_1 + S_2))$ produces the same terms as $(\sum_{\mathbf{x}:Q} S_1) + (\sum_{\mathbf{x}:Q} S_2)$. Like above, the transformation applied to L_1 results in an $LP(CQ_\Sigma)$ L_2 of size at most $d_\Sigma |L_1| \leq d_\Sigma d_\forall |L|$ where d_Σ is the maximal number of enclosed \sum expressions of L .

It remains to observe that a constraint of the form $\forall \mathbf{x}_1 : Q_1 \dots \forall \mathbf{x}_k : Q_k.B$ where B is of the form $S = S'$ or $S \leq S'$ can be rewritten as

$$\forall (\mathbf{x}_1, \dots, \mathbf{x}_k) : (Q_1 \wedge \dots \wedge Q_k).B.$$

This is only true when $set(\mathbf{x}_1), \dots, set(\mathbf{x}_k)$ are pairwise disjoint, which is ensured by the fact that we adopted Barendregt's variables convention. Indeed, we can always rename variables that are bounded by a quantifier so that they have different names. We show it for the case $k = 2$, the general case being a straightforward induction from there.

Let \mathbb{D} be a database and let

- $E = \forall \mathbf{x}_1 : Q_1 \forall \mathbf{x}_2 : Q_2.B$ and
- $F = \forall (\mathbf{x}_1, \mathbf{x}_2) : (Q_1 \wedge Q_2).B,$

By definition, for α mapping every free variables of E and F (they have the same free variables by definition), we have:

$$\begin{aligned} close(E)^{\mathbb{D},\alpha} &= \bigwedge_{\gamma_1 \in \llbracket Q'_1 \rrbracket^{\mathbb{D}}} \bigwedge_{\gamma_2 \in subs_{\gamma_1}(\llbracket Q'_2 \rrbracket^{\mathbb{D}})} close(B)^{\mathbb{D},\tilde{\alpha} \cup \gamma_1 \cup \gamma_2} \\ close(F)^{\mathbb{D},\alpha} &= \bigwedge_{\gamma \in \llbracket Q'_1 \wedge Q'_2 \rrbracket^{\mathbb{D}}} close(B)^{\mathbb{D},\tilde{\alpha} \cup \gamma} \end{aligned}$$

where $Q'_1 = ext_{\mathbf{x}_1}(subs_{\tilde{\alpha}}(Q_1))$ and $Q'_2 = ext_{\mathbf{x}_2}(subs_{\tilde{\alpha}}(Q_2))$

Hence, the proof follows from the fact that $\llbracket Q'_1 \wedge Q'_2 \rrbracket^{\mathbb{D}}$ are the same as the set of γ such that:

- $\gamma|_{set(\mathbf{x}_1)} = \gamma_1$ is in $\llbracket Q'_1 \rrbracket^{\mathbb{D}}$,
- $\gamma|_{set(\mathbf{x}_2)} = \gamma_2$ is such that $\gamma_1 \cup \gamma_2$ is in $\llbracket Q'_2 \rrbracket^{\mathbb{D}}$

which is clear from the definition of conjunctive queries and the fact that $set(\mathbf{x}_1) \cap set(\mathbf{x}_2)$.

Similarly, the closure of linear sums of the form $\sum_{\mathbf{x}_1:Q_1} \cdots \sum_{\mathbf{x}_k:Q_k} S$ will generate the same terms as

$$\sum_{(\mathbf{x}_1, \dots, \mathbf{x}_k):(Q_1 \wedge \cdots \wedge Q_k)} S.$$

The size of L' is then at most $d_\Sigma d_\forall |L| \leq |L|^3$. □

Linear programs in $LP(CQ_\Sigma)$ in normal form allows us to upper bound the size of the closure more precisely. For an $LP(CQ_\Sigma)$ L in normal form, we denote by $\text{Queries}_\forall(L)$ the set of conjunctive query Q that appear in L in an expression of the form $\forall \mathbf{x} : Q$ and by $\text{Queries}_\Sigma(L)$ the set of conjunctive query Q that appear in L in an expression of the form $\sum_{\mathbf{x}:Q}$. As for $LP_{\text{clos}}(CQ_\Sigma)$, we let $\text{Queries}_w(L)$ be the set of conjunctive query Q that appear in L in an expression of the form **weight** $_{\mathbf{z}:\mathbf{x} \doteq \mathbf{y}}(Q)$. We let $\text{AGM}_\forall(L)$ to be $\max_{Q \in \text{Queries}_\forall(L)} \text{AGM}(Q)$ and similarly, $\text{AGM}_\Sigma(L)$ is $\max_{Q \in \text{Queries}_\Sigma(L)} \text{AGM}(Q)$ and $\text{AGM}_w(L)$ is $\max_{Q \in \text{Queries}_w(L)} \text{AGM}(Q)$.

Observe that, given a database \mathbb{D} , an atomic linear constraint of the form $\forall \mathbf{x}:Q. B$ will generate at most $|\mathbb{D}|^{\text{AGM}(Q)}$ constraints in the closure $\text{close}(L)^\mathbb{D}$. Similarly, an atomic linear sum of the form $\sum_{\mathbf{x}:Q} B$ will generate at most $|\mathbb{D}|^{\text{AGM}(Q)}$ terms in $\text{close}(L)^\mathbb{D}$. Hence, we have the following:

Proposition 5.4. *Let L be an $LP(CQ_\Sigma)$ in normal form and \mathbb{D} be a database. $\text{close}(L)^\mathbb{D}$ has at most $|L| \cdot |\mathbb{D}|^{\text{AGM}_\forall(L)}$ constraints, of size at most $|L| \cdot |\mathbb{D}|^{\text{AGM}_\Sigma(L)}$. In particular, $\langle L \rangle^\mathbb{D}$ has at most $|L| \cdot |\mathbb{D}|^{\text{AGM}_\forall(L)}$ constraints, $|L| \cdot |\mathbb{D}|^{\text{AGM}_w(L)}$ variables and can be computed in time $O(|L| \cdot |\mathbb{D}|^{\text{AGM}_\Sigma(L) + \text{AGM}_\forall(L) + \text{AGM}_w(L)})$. Hence, there exists $\ell < 2.37286$ such that one can compute $\text{opt}(\langle L \rangle^\mathbb{D})$ in time $O(|L|^\ell \cdot |\mathbb{D}|^{\ell(\text{AGM}_\Sigma(L) + \text{AGM}_\forall(L) + \text{AGM}_w(L))})$.*

Proposition 5.4 directly implies that $LP(CQ_\Sigma)$ can be solved in polynomial time in the data complexity.

Factorized interpretation of $LP(CQ_\Sigma)$. One can get a better time complexity than the one stated in Proposition 5.4 by exploiting the factorized interpretation presented in Section 4. Indeed, one can directly lift Definition 4.2 of hypertree decomposition and fractional hypertree width of $LP_{\text{clos}}(CQ_\Sigma)$ to $LP(CQ_\Sigma)$. Now, observe that for every $LP(CQ_\Sigma)$ programs for L and every database \mathbb{D} , we have $\text{Queries}_w(L) = \text{Queries}_w(\text{close}(L)^\mathbb{D})$. Hence we have the following:

Lemma 5.5. *Let L be an $LP(CQ_\Sigma)$ program and let \mathcal{T} be a tree decomposition of L . Then for every \mathbb{D} , \mathcal{T} is a tree decomposition of $\text{close}(L)^\mathbb{D}$.*

Hence, let L be an $LP(CQ_\Sigma)$ in normal form and \mathcal{T} a tree decomposition of L of width k . Given a database \mathbb{D} and an $LP(CQ_\Sigma)$ L in normal form, one can compute $L' = \text{close}(L)^\mathbb{D}$ in time $O(|L| \cdot |\mathbb{D}|^{\text{AGM}_\forall(L) + \text{AGM}_\Sigma(L)})$. Then using Theorem 4.9, one can compute $\text{opt}(L')$ in time $O((|L'| + qt|\mathbb{D}|^k)^\ell)$ where q and t respectively denotes the sum of sizes of the queries in $\text{Queries}_w(L)$ and of the tree decompositions in \mathcal{T} . Hence, in the data complexity model, there exists $\ell < 2.37286$ such that one can compute $\text{opt}(\langle L \rangle^\mathbb{D})$ in time $O(|\mathbb{D}|^{\ell p})$ where $p = \max(k, \text{AGM}_\forall(L) + \text{AGM}_\Sigma(L))^5$.

⁵In the conference version of this paper [CCNR22], we were restricting $LP(CQ_\Sigma)$ to only use queries with one atom in $\text{Queries}_\forall(L)$ and $\text{Queries}_\Sigma(L)$ to get a tractable fragment of $LP(CQ_\Sigma)$. Our assumption

While it is not clear to us how one could avoid unfolding every universal quantifiers when constructing the factorized interpretation of an $LP(CQ_\Sigma)$, we explain how one could possibly get a complexity that sometimes may be smaller than $|\mathbb{D}|^{\text{AGM}_\Sigma(L)}$ when computing the closure of a linear expression. Observe that when computing the closure over a database \mathbb{D} of an expression of the form $\sum_{\mathbf{x}:Q} 1$, it will evaluate to the size of $\llbracket Q \rrbracket^{\mathbb{D}}$. Now, if Q has fractional hypertree width k , one can reduce the data complexity of this task from $O(|\mathbb{D}|^{\text{AGM}(Q)})$ to $O(|\mathbb{D}|^k)$ using [PS13]. Similarly, when evaluating expression of the form $\sum_{\mathbf{x}:Q} \mathbf{weight}_{\mathbf{x}':\mathbf{y}=\mathbf{z}}(Q')$, we can first exploit a tree decomposition of Q' of width k' to compute $\llbracket Q' \rrbracket^{\mathbb{D}}_{\text{set}(\mathbf{y})}$ which will be of size at most $|\mathbb{D}|^{k'}$. Now observe that the closure of $\sum_{\mathbf{x}:Q} \mathbf{weight}_{\mathbf{x}':\mathbf{y}=\mathbf{z}}(Q')$ over \mathbb{D} will be of the form $\sum_{\alpha \in \llbracket Q' \rrbracket^{\mathbb{D}}_{\text{set}(\mathbf{y})}} K_\alpha \cdot \mathbf{weight}_{\mathbf{y}=\alpha(\mathbf{y})}(Q')$ where K_α is the number of $\gamma \in \llbracket Q \rrbracket^{\mathbb{D}}$ such that $\gamma(\mathbf{z}) = \alpha(\mathbf{y})$. Again, if Q has bounded fractional hypertree width k , one can compute K_α efficiently since the condition $\gamma(\mathbf{z}) = \alpha(\mathbf{y})$ corresponds into some condition of the values of that some variables in \mathbf{x} have to take. Hence, since computing the list of possible α being doable in time $O(|\mathbb{D}|^{k'})$ and since computing K_α for each α can be done in time $O(|\mathbb{D}|^k)$, we can compute the closure in time $O(|\mathbb{D}|^{k+k'})$ if we have the relevant tree decomposition.

5.3. Case study. The practical performances of our idea heavily depends on how linear solvers perform on factorized interpretation. We compared the performances of GLPK on both the natural interpretation and the factorized interpretation of the resource delivery problem from Section 1.1 using some synthetic data.

For each run we fixed an input size m as well as a domain D of size $n = f(m)$. We then generated each input table of arity k by uniformly sampling m tuples from the n^k possible tuples on D . The value of k was defined so that the ratio of selected tuples $\frac{m}{n^k}$ was constant throughout the runs. We used Python and the Pulp library to build the linear programs as well as a hard-coded tree-decomposition of the *dlr* query (see Section 1.1). The tests were run on an office laptop by progressively increasing the size of the generated input tables. A summary of our experiments is displayed on Figure 10.

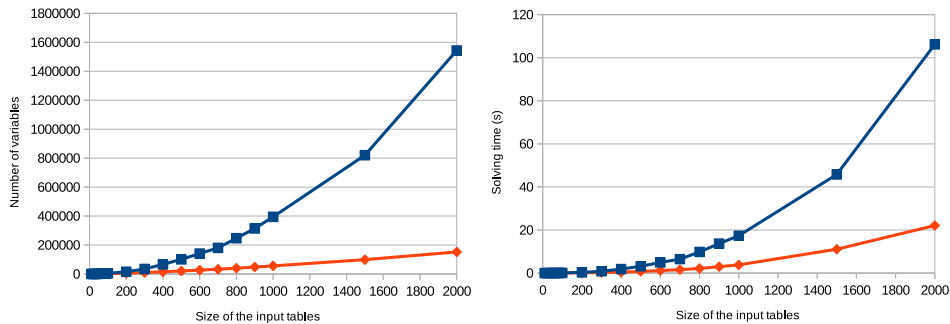


Figure 10: Number of variables and performances of GLPK for natural (blue) and factorized (red) interpretation of the resource delivery problem with respect to table size.

allowed us to bound the size of the closure, which is now implicitly done by observing that in this case, $\text{AGM}_\forall(L) + \text{AGM}_\Sigma(L) \leq 2$. This new formulation allows us to be more precise and provides better bounds.

Interestingly, in this example, the theoretical guarantees given by the factorized interpretation should not be much better than the theoretical guarantees given by the natural interpretation. Indeed, recall that the query considered in this example is the following one:

$$dlr(f, w, b, o) = \exists q. \exists q_2. \exists c \exists c_2. \text{prod}(f, o, q) \wedge \text{order}(b, o, q_2) \wedge \text{route}(f, w, c) \wedge \text{route}(w, b, c_2)$$

Observe that the existentially quantified variables functionally depends on the free variables of dlr . For example, in the table $prod$, q represents the quantity of objects o that factory f can produce, q is hence functionally dependent on f, o . Hence, the AGM bound for dlr implies that for every database \mathbb{D} , $\llbracket dlr \rrbracket^{\mathbb{D}}$ is of size at most $|\mathbb{D}|^2$. Now, observe that the fractional hypertree width of dlr is also 2. Hence, both the factorized interpretation and the natural interpretation may have up to $O(|\mathbb{D}|^2)$ variables. Observe however, that, in the light of Lemma 2.4, the number of variables in the factorized interpretation will never exceed the number of variables in the natural interpretation multiply by a factor depending only on the size of the tree decomposition. But, even if the decomposition has width 2, the factorized interpretation may still be smaller in practice than the natural interpretation and that is what our experiments show. The decomposition we used for the experiments consist in a normalized version of the tree decomposition having two connected vertices u_1, u_2 with $\mathcal{B}(u_1) = \{f, o, q, w, b, c_2\}$ and $\mathcal{B}(u_2) = \{b, o, q_2, f, w, c\}$. Hence, the variables in the factorized interpretation will be the projection of $\llbracket dlr \rrbracket^{\mathbb{D}}$ over $\mathcal{B}(u_1)$ and $\mathcal{B}(u_2)$. It turns out that in the syntactical data we have experimented on, these projections are much smaller than the total size of $\llbracket dlr \rrbracket^{\mathbb{D}}$, leading to a factorized interpretation that is more succinct than the natural one.

As expected when comparing both linear programs we observed a larger number of constraints (due to the soundness constraints) in the factorized interpretation. We observe that the factorized interpretation has less variables, as explained in the previous paragraph. While building the natural interpretation quickly became slower than building the factorized interpretation, we did not analyze this aspect further since we are not using a database engine to build the natural interpretation and solve it directly from the tree decomposition, which may not be the fastest method without further optimizations. Most interestingly solving the factorized interpretation was faster than solving the natural interpretation in spite of the increased number of constraints thanks to the decrease in the number of variables. In particular for an instance with an input size of 2000 lines per table, the natural interpretation had roughly 1.5 million variables while the factorized interpretation had only roughly 150000. The solving time was also noticeably improved at 22s for the factorized case against 106s for the natural one.

6. WEIGHTINGS ON TREE DECOMPOSITIONS

This section is dedicated to the proof of Theorem 4.5 stating the soundness of the factorized interpretation. The soundness of the factorized interpretation boils down to a purely algebraic result concerning conjunctive queries that we now explain. Let Q be a conjunctive query, \mathbb{D} a database, $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ a tree decomposition of Q . We are interested in weightings of $\llbracket Q \rrbracket^{\mathbb{D}}$, that is, mappings $w: \llbracket Q \rrbracket^{\mathbb{D}} \rightarrow \mathbb{R}^+$. Such a mapping naturally defines a mapping $\pi_T(w)$ from $\{\alpha_{|\mathcal{B}(u)} \mid \alpha \in \llbracket Q \rrbracket^{\mathbb{D}}, u \in \mathcal{V}\}$ to \mathbb{R}^+ as follows: for $\beta = \alpha_{|\mathcal{B}(u)}$ for some $u \in \mathcal{V}$, we define the projection of w on T as follows: $\pi_T(w)(\beta) = \sum_{\gamma \in \llbracket Q \rrbracket^{\mathbb{D}}: \gamma_{|\mathcal{B}(u)} = \beta} w(\gamma)$. That is, the weight of β is obtained by summing the weight of every answer of Q compatible with β .

We show in Section 6.1 that the soundness of the factorized interpretation boils down to inverting this projection. Namely, if we are given a weighting W of $\{\alpha_{|\mathcal{B}(u)} \mid \alpha \in \llbracket Q \rrbracket^{\mathbb{D}}, u \in \mathcal{V}\}$, can we construct a weighting $\Pi(W)$ of $\llbracket Q \rrbracket^{\mathbb{D}}$ such that $\pi_T(\Pi(W)) = W$? While this is not always possible, we show in Section 6.2 that it is always possible as long as W is sound, a property that roughly says that $W(\beta)$ and $W(\gamma)$ could not be independent if β and γ are compatible tuples (that is, they assign the same value to their common variables). The proof structure is as follows: Proposition 6.6 established that $\pi_T(w)$ is sound. Then Theorem 6.13 explains the constructions of $\Pi(W)$ from a sound W . The proof of this theorem relies on a good understanding on how projections of the form $\llbracket Q \rrbracket^{\mathbb{D}}_{|\mathcal{B}(u)}$ are related to one another, which is done via several intermediate lemmas presented in Section 6.2.

6.1. Factorized interpretation and weightings. By Proposition 3.5, to prove Theorem 4.5, it is sufficient to show that for every quantifier free conjunctive query Q with tree decomposition $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ and for any set \mathbf{W} over weight expressions over Q , we have $\llbracket \rho^{\mathcal{T}, \mathbb{D}}(\mathbf{W}) \rrbracket_{|\nu(\mathbf{W})} = \llbracket \langle wc^\nu(\mathbf{W}) \rangle^{\mathbb{D}} \rrbracket_{|\nu(\mathbf{W})}$.

In other words, we have to prove the following:

- Lemma 6.1.** • *Given $w_2 \in \llbracket \langle wc^\nu(\mathbf{W}) \rangle^{\mathbb{D}} \rrbracket$, there exists $w_1 \in \llbracket \rho^{\mathcal{T}, \mathbb{D}}(\mathbf{W}) \rrbracket$ such that for every $W \in \mathbf{W}$, $w_1(\nu(W)) = w_2(\nu(W))$.*
 • *Given $w_1 \in \llbracket \rho^{\mathcal{T}, \mathbb{D}}(\mathbf{W}) \rrbracket$, there exists $w_2 \in \llbracket \langle wc^\nu(\mathbf{W}) \rangle^{\mathbb{D}} \rrbracket$ such that for every $W \in \mathbf{W}$, $w_1(\nu(W)) = w_2(\nu(W))$.*

Now recall that $\langle wc^\nu(\mathbf{W}) \rangle^{\mathbb{D}}$ is a set of equality constraints of the form:

$$\nu(\mathbf{weight}_{\mathbf{x}=\mathbf{c}}(Q)) \doteq \sum_{\substack{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x})=\mathbf{c}}} \theta_Q^\alpha.$$

In other words, $w_2 \in \llbracket \langle wc^\nu(\mathbf{W}) \rangle^{\mathbb{D}} \rrbracket$ is a function that maps every θ_Q^α to a value in \mathbb{R}^+ and maps $\nu(\mathbf{weight}_{\mathbf{x}=\mathbf{c}}(Q))$ to $\sum_{\substack{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x})=\mathbf{c}}} w_2(\theta_Q^\alpha)$. We thus have a one-to-one correspondence

between $\llbracket \langle wc^\nu(\mathbf{W}) \rangle^{\mathbb{D}} \rrbracket$ and weightings of $\llbracket Q \rrbracket^{\mathbb{D}}$ by associating $w_2 \in \llbracket \langle wc^\nu(\mathbf{W}) \rangle^{\mathbb{D}} \rrbracket$ to the weighting ω of $\llbracket Q \rrbracket^{\mathbb{D}}$ such that for every $\alpha \in \llbracket Q \rrbracket^{\mathbb{D}}$, $\omega(\alpha) := w_2(\theta_Q^\alpha)$.

Similarly, recall that $\rho^{\mathcal{T}, \mathbb{D}}(\mathbf{W})$ is a set of equality constraints of the form:

$$\nu(\mathbf{weight}_{\mathbf{x}=\mathbf{c}}(Q)) = \xi_{Q,u}^\beta$$

and a conjunction of local soundness constraints, for every edge $e = (u, v) \in \mathcal{E}$ of T and $\gamma \in \llbracket Q \rrbracket^{\mathbb{D}}_{|\mathcal{B}(u) \cap \mathcal{B}(v)}$, we define the equality constraint $E_\gamma^{e, \mathbb{D}}(Q)$ as follows:

$$\sum_{\substack{\beta \in \llbracket Q \rrbracket^{\mathbb{D}}_{|\mathcal{B}(u)} \\ \gamma = \beta_{|\mathcal{B}(u)}}} \xi_{Q,u}^\beta \doteq \sum_{\substack{\beta' \in \llbracket Q \rrbracket^{\mathbb{D}}_{|\mathcal{B}(v)} \\ \gamma = \beta'_{|\mathcal{B}(v)}}} \xi_{Q,v}^{\beta'}.$$

Hence, one can naturally associate $\llbracket \rho^{\mathcal{T}, \mathbb{D}}(\mathbf{W}) \rrbracket$ to a family of weightings $(W_u)_{u \in \mathcal{V}}$ where W_u is a weighting of $\llbracket Q \rrbracket^{\mathbb{D}}_{|\mathcal{B}(u)}$. We do so by associating $w_1 \in \llbracket \rho^{\mathcal{T}, \mathbb{D}}(\mathbf{W}) \rrbracket$ to the family of weightings W_u such that for every $\beta \in \llbracket Q \rrbracket^{\mathbb{D}}_{|\mathcal{B}(u)}$, $W_u(\beta) = w_1(\xi_{Q,u}^\beta)$.

Observe however that every family of $(W_u)_{u \in \mathcal{V}}$ cannot always be mapped back to $\llbracket \rho^{T, \mathbb{D}}(\mathbf{W}) \rrbracket$ since it may not satisfy the local soundness constraints. One needs also $(W_u)_{u \in \mathcal{V}}$ to be *sound*, that is, for every edge $(u, v) \in \mathcal{E}$ and $\gamma \in \llbracket Q \rrbracket^{\mathbb{D}}|_{\mathcal{B}(u) \cap \mathcal{B}(v)}$, we have:

$$\sum_{\substack{\beta \in \llbracket Q \rrbracket^{\mathbb{D}}|_{\mathcal{B}(u)} \\ \gamma = \beta|_{\mathcal{B}(u)}}} W_u(\beta) = \sum_{\substack{\beta' \in \llbracket Q \rrbracket^{\mathbb{D}}|_{\mathcal{B}(v)} \\ \gamma = \beta'|_{\mathcal{B}(v)}}} W_v(\beta').$$

Observe that, when interchanging $\xi_{Q,u}^{\beta}$ and $W_u(\beta)$ in the equality, it exactly corresponds to the local soundness constraints of Q over \mathbb{D} . Hence, we have a one-to-one correspondence between $\llbracket \rho^{T, \mathbb{D}}(\mathbf{W}) \rrbracket$ and sound family of weightings $(W_u)_{u \in \mathcal{V}}$.

Hence, proving Lemma 6.1 boils down to the following. For a query Q , a set of weight expressions \mathbf{W} , a tree decomposition of $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ of Q compatible with \mathbf{W} , we have:

- Given a weighting w of $\llbracket Q \rrbracket^{\mathbb{D}}$, there exists a sound family of weightings $(W_u)_{u \in \mathcal{V}}$ such that for every $\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q) \in \mathbf{W}$, we have:

$$W_u([\mathbf{x}/\mathbf{c}]) = \sum_{\substack{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x}) = \mathbf{c}}} w(\alpha),$$

where u is the vertex of T closest to the root such that $\mathcal{B}(u) = \text{set}(\mathbf{x})$.

- Given a sound family of weightings $(W_u)_{u \in \mathcal{V}}$, there exists a weighting w of $\llbracket Q \rrbracket^{\mathbb{D}}$ such that for every $\mathbf{weight}_{\mathbf{x} \doteq \mathbf{c}}(Q) \in \mathbf{W}$, we have:

$$\sum_{\substack{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x}) = \mathbf{c}}} w(\alpha) = W_u([\mathbf{x}/\mathbf{c}]),$$

where u is the vertex of T closest to the root such that $\mathcal{B}(u) = \text{set}(\mathbf{x})$.

The existence of such weightings will be proven in Theorem 6.13. Proving the first item is actually relatively straightforward as it is sufficient to define $W_u([\mathbf{x}/\mathbf{c}]) = \sum_{\substack{\alpha \in \llbracket Q \rrbracket^{\mathbb{D}} \\ \alpha(\mathbf{x}) = \mathbf{c}}} w(\alpha)$

and prove that it yields a sound weighting. The second item requires a bottom up inductive construction from T . Section 6.2 is dedicated to proving this correspondence between weightings.

6.2. Constructing Weightings. To make notations lighter, we fix in this section a relation $A \subseteq D^X = \{\alpha \mid \alpha : X \rightarrow D\}$ on a finite set of variables X . In this work, A can be thought as the answer set of a conjunctive query Q with $fv(Q) = X$ on database with domain D but the results presented in this section could apply to any relation that is *conjunctive* with respect to a tree decomposition (see Definition 6.7 for more details).

We also fix $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ a decomposition tree for X and define a few useful notations. Given two nodes $u, v \in \mathcal{V}$ we denote the intersection of their bags by $\mathcal{B}^{uv} = \mathcal{B}(u) \cap \mathcal{B}(v)$. We denote by $\downarrow u$ the set of vertices v such that v is in the subtree rooted in u (u included) and by $\uparrow u$ the set of vertices containing u and every vertex v not in $\downarrow u$. We extend the notation: $\mathcal{B}(\downarrow u)$ (resp. $\mathcal{B}(\uparrow u)$) is the union of $\mathcal{B}(v)$ for v in $\downarrow u$ (resp. $\uparrow u$).

6.2.1. *Projections and Extensions.* We start by introducing a few notations to formally restrict relations and manipulate weightings on relations that will be necessary to write down the proofs. Let $X' \subseteq X \subseteq \mathcal{X}$. For any $\alpha' : X' \rightarrow D$ we define the set of its extensions into A by:

$$A[\alpha'] = \{\alpha \in A \mid \alpha|_{X'} = \alpha'\}.$$

For a weighting ω of A and subset of variables $X' \subseteq X$, the projection of ω on X' denoted as $\pi_{X'}(\omega) : A|_{X'} \rightarrow \mathbb{R}^+$ is defined for all $\alpha' \in A|_{X'}$ as:

$$\pi_{X'}(\omega)(\alpha') = \sum_{\alpha \in A[\alpha']} \omega(\alpha).$$

We make a few useful observations on how extensions and projections interact with one another. Formal proofs of these statements may be found in the appendix.

Lemma 6.2. *For any two $\alpha_1, \alpha_2 \in A|_{X'}$, if $\alpha_1 \neq \alpha_2$ then $A[\alpha_1] \cap A[\alpha_2] = \emptyset$.*

Lemma 6.3. *For $A \subseteq D^X$, $X'' \subseteq X' \subseteq X$, $\alpha'' \in A|_{X''}$: $A[\alpha''] = \biguplus_{\alpha' \in A|_{X'}[\alpha'']} A[\alpha']$.*

Lemma 6.4. *For $A \subseteq D^X$, $\omega : A \rightarrow \mathbb{R}^+$, $X'' \subseteq X' \subseteq X$: $\pi_{X''}(\omega) = \pi_{X''}(\pi_{X'}(\omega))$.*

6.2.2. *Weighting Collections.*

Definition 6.5. A family $\Omega = (\Omega_v)_{v \in \mathcal{V}}$ is a *weighting collection on T for A* if it satisfies the following conditions for any two nodes $u, v \in \mathcal{V}$:

- Ω_u is a **weighting of $A|_{\mathcal{B}(u)}$** : i.e., $\Omega_u : A|_{\mathcal{B}(u)} \rightarrow \mathbb{R}^+$.
- Ω_u is **sound for T at $\{u, v\}$** : i.e., $\pi_{\mathcal{B}^{uv}}(\Omega_u) = \pi_{\mathcal{B}^{uv}}(\Omega_v)$.

Intuitively, the soundness of a weighting collection on T is a minimal requirement for the existence of a weighting ω of A such that Ω_u is the projection of ω on the bag $\mathcal{B}(u)$ of T , that is $\Omega_u = \pi_{\mathcal{B}(u)}(\omega)$ since we have the following:

Proposition 6.6. *For any weighting $\omega : A \rightarrow \mathbb{R}^+$, the family $(\pi_{\mathcal{B}(v)}(\omega))_{v \in \mathcal{V}}$ is a weighting collection on T for A .*

Proof. For any $u \in \mathcal{V}$ let $W_u = \pi_{\mathcal{B}(u)}(\omega)$. The first condition on weighting projections holds trivially so we only have to show that the soundness constraint holds. By definition of W_u , $\pi_{\mathcal{B}^{uv}}(W_u) = \pi_{\mathcal{B}^{uv}}(\pi_{\mathcal{B}(u)}(\omega))$. Observe that $\mathcal{B}^{uv} \subseteq \mathcal{B}(u)$ so by Lemma 6.4 $\pi_{\mathcal{B}^{uv}}(W_u) = \pi_{\mathcal{B}^{uv}}(\omega)$. Similarly $\pi_{\mathcal{B}^{uv}}(W_v) = \pi_{\mathcal{B}^{uv}}(\omega)$. \square

What is more interesting is the other way around. For any weighting $\omega : A \rightarrow \mathbb{R}^+$ with $A \subseteq D^X$ and decomposition tree $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ of X , let:

$$\Pi_T(\omega) = (\pi_{\mathcal{B}(v)}(\omega))_{v \in \mathcal{V}}$$

So the question is then given $\Omega = (\Omega_u)_{u \in \mathcal{V}}$ a weighting collection on T whether we can find a weighting ω of A such that $\Omega = \Pi_T(\omega)$. It turns out that soundness is not enough to ensure the existence of such a weighting.

6.2.3. *Conjunctive Decompositions.* However it becomes possible when A is conjunctively decomposed, as we define next. For this, given a decomposition tree $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ and a subsets $V \subseteq \mathcal{V}$ we define $\mathcal{B}(V) = \bigcup_{v \in V} \mathcal{B}(v)$.

Definition 6.7. Let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a decomposition tree of $X \subseteq \mathcal{X}$. We call a subset of variable assignments $A \subseteq D^X$ *conjunctively decomposed by T* if for all $u \in \mathcal{V}$ and $\beta \in A_{|\mathcal{B}(u)}$:

$$\{\alpha_1 \cup \alpha_2 \mid \alpha_1 \in A_{|\mathcal{B}(\uparrow u)}[\beta], \alpha_2 \in A_{|\mathcal{B}(\downarrow u)}[\beta]\} \subseteq A[\beta]$$

Note that the inverse inclusion does hold in general in any case.

Proposition 6.8. *For any tree decomposition T of a quantifier free conjunctive query $Q \in CQ_\Sigma$ and database $\mathbb{D} \in db_\Sigma$, the answer set $\llbracket Q \rrbracket^{\mathbb{D}}$ is conjunctively decomposed by T .*

Proof. Let u be a node of T . Let $R(\mathbf{x})$ be an atom of Q such that $\mathbf{x} \not\subseteq \mathcal{B}(u)$. Then we either have $\mathbf{x} \subseteq \mathcal{B}(\downarrow u)$ or $\mathbf{x} \subseteq \mathcal{B}(\uparrow u)$. Indeed, by definition, there exists v in T such that $\mathbf{x} \subseteq \mathcal{B}(v)$. Since it is not u , v is either in $\mathcal{B}(\downarrow u)$ or $\mathcal{B}(\uparrow u)$ and the result follows. Hence Q can be written as a conjunction $Q_1 \wedge Q_2$ where the variables of Q_1 are included in $\mathcal{B}(\downarrow u)$ and the variables of Q_2 are included in $\mathcal{B}(\uparrow u)$ by defining Q_1 as the set of atom $R(\mathbf{x})$ of Q such that $\mathbf{x} \subseteq \mathcal{B}(\downarrow u)$ and Q_2 to be the other atoms (observe that if $\mathbf{x} \subseteq \mathcal{B}(u)$, $R(\mathbf{x})$ will appear in Q_1 by definition).

Moreover, recall that $\mathcal{B}(u) = \mathcal{B}(\downarrow u) \cap \mathcal{B}(\uparrow u)$ by the connectedness of tree decompositions. Let $\beta \in \llbracket Q \rrbracket^{\mathbb{D}}_{|\mathcal{B}(u)}$ and let $\alpha_1 \in \llbracket Q \rrbracket^{\mathbb{D}}_{|\mathcal{B}(\downarrow u)}[\beta]$ and $\alpha_2 \in \llbracket Q \rrbracket^{\mathbb{D}}_{|\mathcal{B}(\uparrow u)}[\beta]$. We have to show that $\alpha := \alpha_1 \cup \alpha_2 \in \llbracket Q \rrbracket^{\mathbb{D}}[\beta]$. Clearly, $\alpha_{|\mathcal{B}(u)} = \beta$ by construction. It remains to show that $\alpha \in \llbracket Q \rrbracket^{\mathbb{D}}$. To do so, it is sufficient to observe that $\alpha_1 \in \llbracket Q_1 \rrbracket^{\mathbb{D}}$ and $\alpha_2 \in \llbracket Q_2 \rrbracket^{\mathbb{D}}$ which is true since α_1 (and symmetrically α_2) is a projection of some $\alpha' \in \llbracket Q \rrbracket^{\mathbb{D}}$ to $\mathcal{B}(\downarrow u)$ and that the variables of Q_1 are included in $\mathcal{B}(\downarrow u)$. It shows that $\llbracket Q \rrbracket^{\mathbb{D}}$ is conjunctively decomposed by T . \square

Proposition 6.8 does not hold when Q is not quantifier free. It is the reason why this technique only works when every query in the linear program are quantifier free.

Conjunctive decomposition is necessary to get clean relations between the projections of the form $A_{|\mathcal{B}(u)}$ for a vertex u and projections of $A_{|\mathcal{B}(v)}$ for v a child of u . We express these relations depending on the type of u in Lemma 6.9, 6.10, 6.11 and 6.12. These relations will be necessary to prove the correctness of our construction.

Lemma 6.9 (Extend nodes). *Let T be a decomposition tree of X , u an extend node of T with child v , and $A \subseteq D^X$ a subset of variable assignments. If A is conjunctively decomposed by T then any assignment $\beta \in A_{|\mathcal{B}(u)}$ satisfies:*

$$A_{|\mathcal{B}(\downarrow u)}[\beta]_{|\mathcal{B}(\downarrow v)} = A_{|\mathcal{B}(\downarrow v)}[\beta_{|\mathcal{B}(v)}]$$

Proof. For the inclusion from the left to the right let $\alpha \in A_{|\mathcal{B}(\downarrow u)}[\beta]_{|\mathcal{B}(\downarrow v)}$. Since $\alpha \in A_{|\mathcal{B}(\downarrow v)}$ and $\alpha_{|\mathcal{B}(v)} = \beta_{|\mathcal{B}(v)}$ it follows that $\alpha \in A_{|\mathcal{B}(\downarrow v)}[\beta_{|\mathcal{B}(v)}]$.

For the inclusion from the right to the left let $\alpha \in A_{|\mathcal{B}(\downarrow v)}[\beta_{|\mathcal{B}(v)}]$. Let $\gamma \in A_{|\mathcal{B}(\uparrow v)}[\beta]$ be arbitrary and $\tau = \gamma \cup \alpha$.

Note that $(\tau_{|\mathcal{B}(\downarrow u)})_{|\mathcal{B}(\downarrow v)} = \alpha$, so it is sufficient to show $\tau_{|\mathcal{B}(\downarrow u)} \in A_{|\mathcal{B}(\downarrow u)}[\beta]$.

Since u is an extend node with child v it follows that $\mathcal{B}(\uparrow u) = \mathcal{B}(\uparrow v)$, and thus $\gamma \in A_{|\mathcal{B}(\uparrow v)}[\beta]$. By conjunctive decomposition of A by T it follows that $\tau \in A[\beta]$. Hence, $\tau_{|\mathcal{B}(\downarrow u)} \in A_{|\mathcal{B}(\downarrow u)}[\beta]$ as required. \square

Lemma 6.10 (Join nodes). *Let T be a decomposition tree of X , u a join node of T with children v_1, \dots, v_k where $k \geq 1$, and $A \subseteq D^X$ a subset of variable assignments. If A is conjunctively decomposed by T then any $\beta \in A_{|\mathcal{B}(u)}$ satisfies:*

$$A_{|\mathcal{B}(\downarrow u)}[\beta] = A_{|\mathcal{B}(\downarrow v_1)}[\beta] \bowtie \dots \bowtie A_{|\mathcal{B}(\downarrow v_k)}[\beta]$$

Proof. The inclusion from the left to the right is obvious by projecting an element of $A_{|\mathcal{B}(\downarrow u)}[\beta]$ to $\mathcal{B}(\downarrow v_1) \dots \mathcal{B}(\downarrow v_k)$.

For the inclusion from the right to the left let $\alpha_1 \in A_{|\mathcal{B}(\downarrow v_1)}[\beta], \dots, \alpha_k \in A_{|\mathcal{B}(\downarrow v_k)}[\beta]$. We show by induction that $\forall p \in [1, k], \tau_p = \alpha_1 \bowtie \dots \bowtie \alpha_p \in A_{|Y_p}[\beta]$ where $Y_p = \bigcup_{i=1}^p \mathcal{B}(\downarrow v_i)$.

Base case: $p = 1$: Obvious.

Inductive case:: Recall that by induction $\tau_p \in A_{|Y_p}[\beta]$ and observe that $Y_p \subseteq \mathcal{B}(\uparrow v_{p+1})$ so there exists $\gamma \in \mathcal{B}(\uparrow v_{p+1})[\beta]$ such that $\gamma|_{Y_p} = \tau_p$.

By conjunctive decomposition on v_{p+1} , $\alpha = \gamma \bowtie \alpha_{p+1} \in A$. Finally we have $\alpha|_{Y_{p+1}} = (\gamma \bowtie \alpha_{p+1})|_{Y_p \cup \mathcal{B}(\downarrow v_{p+1})} = \gamma|_{Y_p} \bowtie \alpha_{p+1}|_{\mathcal{B}(\downarrow v_{p+1})} = \tau_p \bowtie \alpha_{p+1} = \tau_{p+1}$ so $\tau_{p+1} \in A_{|Y_{p+1}}$. Thus $\tau_{p+1} \in Y_p[\beta]$ because $\tau_{p+1}|_{\mathcal{B}(u)} = \beta$. □

Lemma 6.11. *Let T be a decomposition tree of X and u a join of T with children v_1, \dots, v_k . If $A \subseteq D^X$ is conjunctively decomposed by T then for any $\alpha \in A_{|\mathcal{B}(\downarrow v_1)}$ and $\beta = \alpha|_{\mathcal{B}(u)}$:*

$$A_{|\mathcal{B}(\downarrow u)}[\alpha] = \{\alpha\} \bowtie A_{|\mathcal{B}(\downarrow v_2)}[\beta] \bowtie \dots \bowtie A_{|\mathcal{B}(\downarrow v_k)}[\beta]$$

Proof. Clearly $A_{|\mathcal{B}(\downarrow u)}[\alpha] = \{\tau \in A_{|\mathcal{B}(\downarrow u)}[\beta] \mid \tau|_{\mathcal{B}(\downarrow v_1)} = \alpha\}$ since $\beta = \alpha|_{\mathcal{B}(u)}$. Thus by Lemma 6.10, $A_{|\mathcal{B}(\downarrow u)}[\alpha] = \{\tau \in A_{|\mathcal{B}(\downarrow v_1)}[\beta] \bowtie \dots \bowtie A_{|\mathcal{B}(\downarrow v_k)}[\beta] \mid \tau|_{\mathcal{B}(\downarrow u)} = \alpha\} = \{\alpha\} \bowtie A_{|\mathcal{B}(\downarrow v_2)}[\beta] \bowtie \dots \bowtie A_{|\mathcal{B}(\downarrow v_k)}[\beta]$. □

Lemma 6.12 (Extend node). *Let v be the child of an extend node u . It holds for all $\alpha \in A_{|\mathcal{B}(\downarrow v)}$ with $\beta = \alpha|_{\mathcal{B}(u)}$ that:*

$$A[\alpha] = \biguplus_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} A[\alpha \cup \beta']$$

Proof. For the inclusion from the left to the right, let $\tau \in A[\alpha]$ and $\beta' = \tau|_{\mathcal{B}(u)}$. Observe that $\beta' \in A_{|\mathcal{B}(u)}[\beta]$. Moreover $\mathcal{B}(\downarrow u) = \mathcal{B}(\downarrow v) \cup \mathcal{B}(u)$ so $\tau|_{\mathcal{B}(\downarrow u)} = \alpha \cup \beta'$ so $\tau \in A[\alpha \cup \beta']$.

For the inclusion from the right to the left, let $\tau \in \biguplus_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} A[\alpha \cup \beta']$. By definition $\tau \in A$ and $\tau|_{\mathcal{B}(\downarrow v)} = \alpha$ so $\tau \in A[\alpha]$. □

6.2.4. *Correspondence Theorem.* We can now establish the correspondence.

Theorem 6.13 (Correspondence). *Let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a normalized decomposition tree of $X \subseteq \mathcal{X}$ and $A \subseteq D^X$ be a set of variable assignments that is conjunctively decomposed by T .*

- *For every weighting ω of A , $\Pi_T(\omega)$ is a weighting collection on T for A .*
- *For any weighting collection Ω on T for A there exists a weighting ω of A such that $\Omega = \Pi_T(\omega)$.*

Proof. The first property was shown in Proposition 6.6. So it remains to prove the second property. Let $\Omega = (\Omega_u)_{u \in \mathcal{V}}$ be a weighting collection on T for $A \subseteq D^X$. We start with the construction of ω from $(\Omega_u)_{u \in \mathcal{V}}$. For this we inductively construct for any node $u \in \mathcal{V}$, a weighting $\omega_u : A_{|\mathcal{B}(\downarrow u)} \rightarrow \mathbb{R}^+$, always assuming that $\omega_{u'}$ is defined for all children u' of u . For any $\alpha \in A_{|\mathcal{B}(\downarrow u)}$ and $\beta = \alpha_{|\mathcal{B}(u)}$ we define $\omega_u(\alpha)$ as follows:

Case u is a leaf of T .: We define $\omega_u(\alpha) = \Omega_u(\alpha)$.

Case u is an extend node of T with a single child v .: We define:

$$\omega_u(\alpha) = \begin{cases} \frac{\Omega_u(\beta)}{\Omega_v(\alpha_{|\mathcal{B}(v)})} \omega_v(\alpha_{|\mathcal{B}(\downarrow v)}) & \text{if } \Omega_v(\alpha_{|\mathcal{B}(v)}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Case u is a project node of T with a single child v .: We define $\omega_u(\alpha) = \omega_v(\alpha_{|\mathcal{B}(\downarrow v)})$.

Case u is a join node of T with children v_1, \dots, v_k .: Then we define:

$$\omega_u(\alpha) = \begin{cases} \frac{\prod_{i=1}^k \omega_{v_i}(\alpha_{|\mathcal{B}(\downarrow v_i)})}{\Omega_u(\beta)^{k-1}} & \text{if } \Omega_u(\beta) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Finally, we define $\omega = \omega_r$ where r is the root of T . The proof that $\forall u : \Omega_u = \pi_{\mathcal{B}(u)}(\omega)$ is done via two inductions. The first one is a bottom-up induction to prove that $\Omega_u = \pi_{\mathcal{B}(u)}(\omega_u)$ for every node u in the tree decomposition, see Lemma 6.14 below. Then, by top-down induction, one can prove that $\omega_u = \pi_{\mathcal{B}(\downarrow u)}(\omega_r)$. See Lemma 6.15 below. Thus:

$$\Omega_u = \pi_{\mathcal{B}(u)}(\omega_u) = \pi_{\mathcal{B}(u)}(\pi_{\mathcal{B}(\downarrow u)}(\omega_r)) = \pi_{\mathcal{B}(u)}(\omega_r) = \pi_{\mathcal{B}(u)}(\omega)$$

In other words, $\Omega = \Pi_T(\omega)$ as stated by the proposition. \square

Lemma 6.14. *For all $u \in \mathcal{V}$: $\Omega_u = \pi_{\mathcal{B}(u)}(\omega_u)$.*

Proof. We show by bottom-up induction on the nodes of T that for all $u \in \mathcal{V}$ and $\beta \in A_{|\mathcal{B}(u)}$, $\sum_{\alpha \in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_u(\alpha) = \Omega_u(\beta)$.

The base case is clearly true by the definition of ω_u when u is a leaf.

Case 1: u is an extend node with v its only child.

Let $\beta \in A_{|\mathcal{B}(u)}$ and $\beta' = \beta_{|\mathcal{B}(v)}$.

Case 1.1: $\Omega_v(\beta') = 0$.

By definition $\forall \alpha \in A_{|\mathcal{B}(\downarrow u)}[\beta], \omega_u(\alpha) = 0$.

Recall that by soundness $\sum_{\beta'' \in A_{|\mathcal{B}(u)}[\beta']} \Omega_u(\beta'') = \Omega_v(\beta') = 0$. Observe that

$$\beta \in A_{|\mathcal{B}(u)}[\beta'] \text{ so } \Omega_u(\beta) = 0 = \sum_{\alpha \in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_u(\alpha).$$

Case 1.2: $\Omega_v(\beta') > 0$.

$$\begin{aligned} & \sum_{\alpha \in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_u(\alpha) \\ &= \sum_{\alpha \in A_{|\mathcal{B}(\downarrow u)}[\beta]} \frac{\Omega_u(\beta)}{\Omega_v(\beta')} \omega_v(\alpha_{|\mathcal{B}(\downarrow v)}) && \text{by definition} \\ &= \frac{\Omega_u(\beta)}{\Omega_v(\beta')} \sum_{\alpha \in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_v(\alpha_{|\mathcal{B}(\downarrow v)}) \\ &= \frac{\Omega_u(\beta)}{\Omega_v(\beta')} \sum_{\alpha' \in A_{|\mathcal{B}(\downarrow v)}[\beta']} \omega_v(\alpha') && \text{conj. decomp. Lemma 6.9} \\ & && \text{for extend nodes} \\ &= \frac{\Omega_u(\beta)}{\Omega_v(\beta')} \Omega_v(\beta') && \text{by induction} \\ &= \Omega_u(\beta) \end{aligned}$$

Case 2: u is a project node with only child v .

$$\begin{aligned}
 & \sum_{\alpha \in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_u(\alpha) \\
 &= \sum_{\alpha \in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_u(\alpha) && \text{by definition} \\
 &= \sum_{\beta' \in A_{|\mathcal{B}(v)}[\beta]} \sum_{\alpha' \in A_{|\mathcal{B}(\downarrow u)}[\beta']} \omega_v(\alpha') && \text{by Lemma 6.4 and } \mathcal{B}(v) \subseteq \mathcal{B}(\downarrow u) \\
 &= \sum_{\beta' \in A_{|\mathcal{B}(v)}[\beta]} \Omega_v(\beta') && \text{by induction and } \mathcal{B}(\downarrow u) = \mathcal{B}(\downarrow v) \\
 &= \Omega_u(\beta) && \text{by soundness at } (u, v)
 \end{aligned}$$

Case 3: u is a join node with children v_1, \dots, v_k .

Let $\beta \in A_{|\mathcal{B}(u)}$.

Case 3.1:

$$W_u(\beta) = 0.$$

By definition $\forall \alpha \in A_{|\mathcal{B}(\downarrow u)}[\beta], \omega_u(\alpha) = 0$.

Thus $\sum_{\alpha \in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_u(\alpha) = 0 = \Omega_u(\beta)$.

Case 3.2: $\Omega_u(\beta) > 0$.

$$\begin{aligned}
 & \sum_{\alpha \in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_u(\alpha) \\
 &= \sum_{\alpha \in A_{|\mathcal{B}(\downarrow u)}[\beta]} \frac{\prod_{i=1}^k \omega_{v_i}(\alpha_{|\mathcal{B}(\downarrow v_i)})}{\Omega_u(\beta)^{k-1}} && \text{by definition} \\
 &= \sum_{\alpha_1 \in A_{|\mathcal{B}(\downarrow v_1)}[\beta]} \cdots \sum_{\alpha_k \in A_{|\mathcal{B}(\downarrow v_k)}[\beta]} \frac{\prod_{i=1}^k \omega_{v_i}(\alpha_i)}{\Omega_u(\beta)^{k-1}} && \text{conj. decomp. Lemma 6.10} \\
 & && \text{for join nodes} \\
 &= \frac{\prod_{i=1}^k \sum_{\alpha_i \in A_{|\mathcal{B}(\downarrow v_i)}[\beta]} \omega_{v_i}(\alpha_i)}{\Omega_u(\beta)^{k-1}} \\
 &= \frac{\prod_{i=1}^k \Omega_{v_i}(\beta)}{\Omega_u(\beta)^{k-1}} && \text{by induction} \\
 &= \frac{\Omega_u(\beta)^k}{\Omega_u(\beta)^{k-1}} && \text{by soundness at } (u, v_i) \\
 &= \Omega_u(\beta)
 \end{aligned}$$

□

Lemma 6.15. For all $u \in \mathcal{V}$: $\omega_u = \pi_{\mathcal{B}(\downarrow u)}(\omega_r)$.

Proof. We show by top-down induction on the nodes of T that for all $v \in \mathcal{V}$ and $\alpha \in A_{|\mathcal{B}(\downarrow v)}$, $\sum_{\tau \in A[\alpha]} \omega_r(\tau) = \omega_v(\alpha)$.

The base case is clearly true when v is the root r of T .

In the following we consider a given $\alpha \in A_{|\mathcal{B}(\downarrow v)}$. and we let $\beta = \alpha_{|\mathcal{B}(v)}$

Case 1: v is the only child of an extend node u .

By Lemma 6.12, $\sum_{\tau \in A[\alpha]} \omega_r(\tau) = \sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} \sum_{\tau \in A[\alpha \cup \beta']} \omega_r(\tau)$. By induction this is equal to $\sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} \omega_u(\alpha \bowtie \beta')$.

Case 1.1: $\Omega_v(\beta) = 0$.

By definition of ω_u , $\sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} \omega_u(\alpha \bowtie \beta') = 0$.

Observe that by Proposition 6.14, $\sum_{\alpha' \in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_v(\alpha') = \Omega_v(\beta) = 0$. However

ω_v is non-negative so $\omega_v(\alpha) = 0 = \sum_{\tau \in A[\alpha]} \omega_r(\tau)$.

Case 1.2: $\Omega_v(\beta) > 0$.

$$\begin{aligned}
& \sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} \omega_u(\alpha \bowtie \beta') \\
&= \sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} \frac{\Omega_u(\beta')}{\Omega_v(\beta)} \omega_v((\alpha \bowtie \beta')|_{\mathcal{B}(\downarrow v)}) \quad \text{by definition} \\
&= \frac{\sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} \Omega_u(\beta')}{\Omega_v(\beta)} \omega_v(\alpha) \\
&= \omega_v(\alpha) \quad \text{by soundness at } (u, v)
\end{aligned}$$

Case 2: v is the only child of a project node u .

Observe that $\mathcal{B}(\downarrow u) = \mathcal{B}(\downarrow v)$ because u is a project node so by induction:

$$\sum_{\tau \in A[\alpha]} \omega_r(\tau) = \omega_u(\alpha) = \omega_v(\alpha).$$

Case 3: v is the child of a join node u .

Let v_1, \dots, v_n be the children of u , we assume wlog that v is v_1 .

By Lemma 6.4, $\sum_{\tau \in A[\alpha]} \omega_r(\tau) = \sum_{\alpha' \in A_{|\mathcal{B}(\downarrow u)}[\alpha]} \sum_{\tau \in A[\alpha']} \omega_r(\tau)$.

By induction we obtain $\sum_{\alpha' \in A_{|\mathcal{B}(\downarrow u)}[\alpha]} \omega_u(\alpha')$.

Case 3.1: $\Omega_u(\beta) = 0$.

By definition of ω_u , $\sum_{\alpha' \in A_{|\mathcal{B}(\downarrow u)}[\alpha]} \omega_u(\alpha') = 0$.

Recall that because u is a join node, $\Omega_v(\beta) = \Omega_u(\beta) = 0$ so similarly to Case 1.2, $\omega_v(\alpha) = 0 = \sum_{\tau \in A[\alpha]} \omega_r(\tau)$.

Case 3.2: $\Omega_v(\beta) > 0$.

By definition of ω_u , $\sum_{\alpha' \in A_{|\mathcal{B}(\downarrow u)}[\alpha]} \omega_u(\alpha') = \sum_{\alpha' \in A_{|\mathcal{B}(\downarrow u)}[\alpha]} \frac{\prod_{i=1}^k \omega_{v_i}(\alpha'|_{\mathcal{B}(\downarrow v_i)})}{\Omega_u(\beta)^{k-1}}$. Moreover by Lemma 6.11 we can split α' into $\alpha \times \alpha_2 \times \dots \times \alpha_k$ and the sum into

$\sum_{\alpha_2 \in A_{|\mathcal{B}(\downarrow v_2)}[\beta]} \dots \sum_{\alpha_k \in A_{|\mathcal{B}(\downarrow v_k)}[\beta]} \frac{\prod_{i=1}^k \omega_{v_i}(\alpha'|_{\mathcal{B}(\downarrow v_i)})}{\Omega_u(\beta)^{k-1}}$. Observe that each term in the product only depends on α_i (or α for $i = 1$) and that the denominator only depends on the fixed β so we can rewrite the formula into the following

$\omega_u(\alpha) \cdot \frac{\prod_{i=2}^k \sum_{\alpha_i \in A_{|\mathcal{B}(\downarrow v_i)}[\beta]} \omega_{v_i}(\alpha_i)}{\Omega_u(\beta)^{k-1}}$ which is equal to $\omega_v(\alpha) \cdot \frac{\prod_{i=2}^k \Omega_{v_i}(\beta)}{\Omega_u(\beta)^{k-1}}$ by Proposition 6.14. Finally observe that by soundness, $\prod_{i=2}^k \Omega_{v_i}(\beta) = \Omega_u(\beta)^{k-1}$.

Thus $\sum_{\tau \in A[\alpha]} \omega_r(\tau) = \omega_v(\alpha)$.

□

7. APPLICATIONS

In this section, we provide two applications of our results for existing optimization problems in different areas of computer science.

7.1. Minimizing Noise for ε -Differential Privacy. The strategy of differential privacy is to add noise to the relational data before publication. Roughly speaking, the general objective of ε -differential privacy [DR14] is to add as little noise as possible, without disclosing more than an ε amount of information. We illustrate this with the example of a set of hospitals which publish medical studies aggregating results of tests on patients, which are to be kept confidential. We consider the problem of how to compute the optimal amount of noise to be added to each separate piece of sensitive information (in terms of total utility of the

studies) while guaranteeing ε -differential privacy. We show that this question can be solved (approximately) by computing the optimal solution of a projecting program in $LP(CQ_\Sigma)$ with a single conjunctive query that is acyclic, i.e., of hypertree with 1. While the natural interpretation yields a linear program with a quadratic number of variables in the size of the database, the factorized interpretation requires only a linear number.

We consider a database \mathbb{D} with signature $\Sigma = \{H, Test, St, Priv, Sens\}$ whose domain provides patients, hospitals, studies, and positive real numbers. The relations of \mathbb{D} are the following:

- $(pat, hosp) \in H^{\mathbb{D}}$: the patient pat is in the hospital $hosp$.
- $(pat, st) \in Test^{\mathbb{D}}$: the patient pat participates in the study st .
- $(test, st) \in St^{\mathbb{D}}$: the test $test$ is used in the study st .
- $(obj, \varepsilon) \in Priv^{\mathbb{D}}$: the object obj is either a patient or a hospital. The positive real number ε indicates the privacy budget for obj .
- $(st, test, val) \in Sens^{\mathbb{D}}$: the value (in terms of study results) of a patient participating in a study and contributing a unit of information on their result on test $test$.

The following query defines the sensitive information that will be revealed to the researchers performing the medical studies. It selects all pairs of patients pat and tests $test$, such pat did the $test$ which was then used by some study st .

$$InStudy(pat, test) = \exists st. Test(pat, test) \wedge St(test, st)$$

More precisely, the sensitive information is the answer set of this query over the database \mathbb{D} . We want to assign a weight to all the pairs in the answer set. The weight of a sensitive pair states the amount of information that may be disclosed about the pair after the addition of the noise. The needed amount of noise for the pair is then inversely proportional to the amount of information that may be disclosed, i.e, the weight of the pair, which is also called its privacy budget. The weight of a patient pat and a test $test$ is specified by the weight expression:

$$\mathbf{weight}_{(pat', test') : test' \doteq test \wedge pat' \doteq pat}(InStudy(pat', test'))$$

In an environment γ for the global variables pat and $test$ this weight expression is interpreted as the linear program variable:

$$\theta_{InStudy(pat', test')}^{[pat'/\gamma(pat), test'/\gamma(test)]}$$

The overall weight of all sensitive tests of the same patient pat is described by the weight expression:

$$\mathbf{weight}_{(pat', test') : pat' \doteq pat}(InStudy(pat', test'))$$

In an environment γ for the global variable pat this weight expression is interpreted as the following sum of linear program variables:

$$\sum_{\alpha \in sol^{\mathbb{D}}(InStudy(pat', test') \wedge pat' = \gamma(pat))} \theta_{InStudy(pat', test')}^{[pat'/\gamma(pat), test'/\alpha(test)]}$$

This sum may be represented more compactly in factorized interpretation avoiding the enumeration of the answer set for the database \mathbb{D} .

The $LP(CQ_\Sigma)$ program for this example is given in Figure 11.

The linear privacy constraints that are to be satisfied are C_{PAT} and C_{HOSP} . Constraint C_{PAT} states that for all patients pat with privacy requirement ε , i.e., $\forall(pat, \varepsilon) : Priv(pat, \varepsilon)$, the sum of all weights of all sensitive pairs $(pat, test')$ in $InStudy$ must be bounded by ε .

$$\begin{array}{c}
\text{Queries} \\
InStudy(pat, test) = \exists st. Test(pat, test) \wedge St(st, test) \\
\text{Constraints} \\
C_{PAT} = \forall (pat, \varepsilon): Priv(pat, \varepsilon). \\
\mathbf{weight}_{(pat', test'): pat' \doteq pat} (InStudy(pat', test')) \leq \mathbf{num}(\varepsilon) \\
C_{HOSP} = \forall (hosp, \varepsilon): Priv(hosp, \varepsilon). \sum_{(pat): H(pat, hosp)} \\
\mathbf{weight}_{(pat', test'): pat' \doteq pat} (InStudy(pat', test')) \leq \mathbf{num}(\varepsilon) \\
\text{Program} \\
\mathbf{maximize} \sum_{(st, test, val): Sens(st, test, val)} \\
\mathbf{num}(val) \mathbf{weight}_{(pat', test'): test' \doteq test} (InStudy(pat', test')) \\
\mathbf{subject to} C_{PAT} \wedge C_{HOSP}
\end{array}$$

Figure 11: An $LP^{proj}(CQ_{\Sigma})$ program for differential privacy when publishing medical studies aggregating patient tests in hospitals.

This constraint is motivated by the composition rule of differential privacy (DP). Suppose we have sensitive pairs $p_i = (pat_i, test_i)$. If p_i is ε_i -DP for $1 \leq i \leq n$, then $\{p_1 \dots p_n\}$ is $(\sum_{i=1}^n \varepsilon_i)$ -DP.

Similarly, constraint C_{HOSP} states that for all hospitals $hosp$ with privacy requirement ε , i.e., $\forall (hosp, \varepsilon) : Priv(hosp, \varepsilon)$, the sum of all weights of all sensitive pairs $(pat, test)$ in $InStudy$ where pat is a patient of $hosp$ must be bounded by ε . Finally, the objective function is to maximize the sum over all triples $(st, test, val)$ in $Sens$ of the weights of pairs $(pat', test)$ in $InStudy$ but multiplied with $\mathbf{num}(val)$, the utility of the information for the study.

This program is projecting, so it is a member of $LP^{proj}(CQ_{\Sigma})$. Furthermore, a hypertree decomposition of width 1 is available. While the natural interpretation over a database yields a linear program with a quadratic number of variables (in the size of the database), the factorized interpretation yields a linear program with a linear number of variables.

Please note that the approach presented above is only approximate. For example, summing over noise variance in the objective function would be more accurate but would only lead to a convex program, which motivates us to extend beyond linear programs in future work. Also, the composition rule for DP is only approximate, more advanced composition rules have been studied but they are more complex and still approximate.

7.2. Computing the s-Measure for Graph Pattern Matching. A matching of a subgraph pattern in a graph is a graph homomorphism from the pattern to the graph. The s -measure of Wang et al. [WRF13] is used in data mining to measure the frequency of matchings of subgraph patterns, while accounting for overlaps of different matchings. The idea is to find a maximal weighting for the set of matchings, such that for any node of the subgraph pattern, the set of matchings mapping it on the same graph node must have an overall weight less than 1. This optimization problem can be expressed by a projecting $LP(CQ_{\Sigma})$ program over a database storing the graph. The conjunctive query of this program expresses the matching of the subgraph pattern. The hypertree width of this conjunctive query is bounded by the hypertree width of the subgraph pattern. Our factorized

interpretation therefore reduces the size of the linear program for subgraph patterns with small hypertree width.

The s -measure has been introduced by Wang et al. [WRF13] to evaluate the frequency of matchings of a subgraph pattern in a larger graph. Here, we consider pattern matches as graph homomorphism, but we could also restrict them to graph isomorphisms.

A naive way of evaluating this frequency is to use the number of pattern matches as the frequency measure. Using this value as a frequency measure is problematic since different pattern matches may overlap, and as such they share some kind of dependencies that is relevant from a statistical point of view. More importantly, due to the overlaps, this measure fails to be anti-monotone, meaning that a subpattern may be counter-intuitively matched less frequently than the pattern itself. Therefore, the finding of better anti-monotonic frequency measures – also known as *support measures* – has received a lot of attention in the data mining community [BN08, CRVD11, FB07]. A first idea is to count the maximal number of non-overlapping patterns [VGS02]. However, finding such a maximal subset of patterns essentially boils down to finding a maximal independent set in a graph, a notorious NP-complete problem [GJ79].

The s -measure is a relaxation of this idea where the frequency of pattern matches is computed as the maximum of the sum of the weights that can be assigned to each pattern match, under the constraint that for any node v of the graph and node v' in the subgraph pattern that the sum of the weights of the matchings mapping v' to v is at most 1. More formally, given two digraphs $G = (V_G, E_G)$ and $P = (V_P, E_P)$, we define a matching of the pattern P in graph G as a graph homomorphism $h : V_P \rightarrow V_G$. Recall that a graph homomorphism requires for all $(v, v') \in E_P$ that $(h(v), h(v')) \in E_G$. We denote by $hom(P, G)$ the set of matchings of P in G . The s -measure of P in G is then defines as the optimal value of the following linear program with variables in $\{\theta_h \mid h \in hom(P, G)\}$ for positive real numbers:

$$\begin{aligned} & \mathbf{maximize} && \sum_{h \in hom(P, G)} \theta_h \\ & \mathbf{subject\ to} && \forall v \in V_G. \forall v' \in V_P. \sum_{\substack{h \in hom(P, G) \\ h(v')=v}} \theta_h \leq 1 \end{aligned}$$

We can consider each graph G as a database \mathbb{D} with signature $\Sigma = \{node, edge\}$, domain $D(\mathbb{D}_G) = V_G$ and relations $node^{\mathbb{D}} = V_G$ and $edge^{\mathbb{D}} = E_G$. Since the names of the nodes of the pattern do not care for pattern matching, we can assume without loss of generality that $V_P = \{1, \dots, \ell\}$ for some $\ell \in \mathbb{N}$. We can then define a matching of a pattern P by a conjunctive query $match_P(x_1, \dots, x_\ell)$:

$$match_P(x_1, \dots, x_\ell) = \bigwedge_{(i,j) \in E_P} edge(x_i, x_j)$$

It is clear that $\alpha \in sol^{\mathbb{D}}(match_P(x_1, \dots, x_\ell))$ if and only if $\alpha \circ [1/x_1, \dots, \ell/x_\ell]$ is a pattern matching in $hom(P, Q)$. One can thus rewrite the previous linear program as $LP(CQ_\Sigma)$ program as follows:

$$\begin{aligned} & \mathbf{maximize} && \sum_{(x):node(x)} \mathbf{weight}_{(x_1, \dots, x_n):x_1 \doteq x}(match_P(x_1, \dots, x_n)) \\ & \mathbf{subject\ to} && \forall (x) : node(x) : \bigwedge_{i=1}^{\ell} \mathbf{weight}_{(x_1, \dots, x_n):x_i \doteq x}(match_P(x_1, \dots, x_n)) \leq 1. \end{aligned}$$

Moreover, the hypertree width of the conjunctive query $match_P(x_1, \dots, x_\ell)$ is at most the (hyper)tree width of the pattern graph P . By our main Theorem 4.5, the factorized interpretation yields a linear program with at most $(|V_G| + |E_G|)^k$ variables, where k is the (hyper)tree width of pattern P . The original linear program could have been of size

$\binom{|V_G|}{\ell}$ which is bounded by $|V_G|^\ell$. So the factorized interpretation will pay off if the (hyper)tree width k of the pattern is considerably smaller than the number ℓ of its nodes.

8. CONCLUSION AND FUTURE WORK

In this paper we studied linear programs whose variables are the answers to conjunctive queries. While in general it is possible to construct in this way large and hard to solve programs, we defined a tractable fragment which can benefit from factorization. Our experiments suggest the efficiency of factorized interpretation, in accordance with our complexity results.

Several directions for future work exist. One direction is to explore how to better integrate our approach into a database engine, in the way it is done by `SolveDB` for example. Also, other optimization problems may benefit from this approach such as convex optimization or integer linear programming. It would be interesting to define languages analogous to $LP(CQ_\Sigma)$ for these optimization problems and study how conjunctive query decompositions could help to improve the efficiency.

REFERENCES

- [AGM13] Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013. doi:10.1137/110859440.
- [AW21] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.
- [Bar12] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics. College Publications, 2012. doi:10.2307/2274112.
- [BDG07] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *International Workshop on Computer Science Logic*, pages 208–222. Springer, 2007. doi:10.1007/978-3-540-74915-8_18.
- [BN08] Björn Bringmann and Siegfried Nijssen. What is frequent in a single graph? In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 858–863. Springer, 2008. doi:10.1007/978-3-540-68125-0_84.
- [CCNR22] Florent Capelli, Nicolas Crosetti, Joachim Niehren, and Jan Ramon. Linear programs with conjunctive queries. In Dan Olteanu and Nils Vortmeier, editors, *25th International Conference on Database Theory, ICDT 2022, March 29 to April 1, 2022, Edinburgh, UK (Virtual Conference)*, volume 220 of *LIPICs*, pages 5:1–5:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICDT.2022.5.
- [CLS21] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39, 2021. doi:10.1145/3424305.
- [CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, STOC '77*, pages 77–90, New York, NY, USA, 1977. ACM. doi:10.1145/800105.803397.
- [CRVD11] Toon Calders, Jan Ramon, and Dries Van Dyck. All normalized anti-monotonic overlap graph measures are bounded. *Data Mining and Knowledge Discovery*, 23(3):503–548, 2011. doi:10.1007/s10618-011-0217-y.
- [DR14] Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014. doi:10.1561/04000000042.
- [FB07] Mathias Fiedler and Christian Borgelt. Support computation for mining frequent subgraphs in a single graph. In *MLG*. Citeseer, 2007.
- [FGK90] Robert Fourer, David M Gay, and Brian W Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, 1990. doi:10.1287/mnsc.36.5.519.

- [GJ79] Michael R Garey and David S Johnson. *Computers and intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman And Company, 1979.
- [GLS99] Georg Gottlob, Nicola Leone, and Francesco Scarcello. On tractable queries and constraints. In *International Conference on Database and Expert Systems Applications*, pages 1–15. Springer, 1999. doi:10.1007/3-540-48309-8_1.
- [GLS02] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree Decompositions and Tractable Queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002. doi:10.1145/303976.303979.
- [GM14] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms (TALG)*, 11(1):4, 2014. doi:10.1145/2636918.
- [Gro06] Martin Grohe. The structure of tractable constraint satisfaction problems. In *International Symposium on Mathematical Foundations of Computer Science*, pages 58–72. Springer, 2006. doi:10.1007/11821069_5.
- [Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984. doi:10.1145/800057.808695.
- [Klo94] Ton Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994.
- [KPT13] Phokion G. Kolaitis, Enela Pema, and Wang-Chiew Tan. Efficient querying of inconsistent databases with binary integer programming. *Proceedings of the VLDB Endowment*, 6(6):397–408, 2013. doi:10.14778/2536336.2536341.
- [Lib13] Leonid Libkin. *Elements of finite model theory*. Springer Science & Business Media, 2013. doi:10.1007/978-3-662-07003-1.
- [MS12] Alexandra Meliou and Dan Suciu. Tiresias: The database oracle for how-to queries. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, SIGMOD '12*, pages 337–348, New York, NY, USA, 2012. ACM. doi:10.1145/2213836.2213875.
- [NSB⁺07] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007. doi:10.1007/978-3-540-74970-7_38.
- [OZ12] Dan Olteanu and Jakub Závodný. Factorised representations of query results: size bounds and readability. In *Proceedings of the 15th International Conference on Database Theory*, pages 285–298. ACM, 2012. doi:10.1145/2274576.2274607.
- [OZ15] Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Transactions on Database Systems (TODS)*, 40(1):1–44, 2015. doi:10.1145/2693969.
- [PS13] Reinhard Pichler and Sebastian Skritek. Tractable counting of the answers to conjunctive queries. *Journal of Computer and System Sciences*, 79:984–1001, 2013. doi:10.1016/j.jcss.2013.01.012.
- [ŠP16] Laurynas Šikšnyš and Torben Bach Pedersen. SolveDB: Integrating optimization problem solvers into SQL databases. In *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*, page 14. ACM, 2016. doi:10.1145/2949689.2949693.
- [Vel14] Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy, editors, *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*, pages 96–106. OpenProceedings.org, 2014. doi:10.5441/002/icdt.2014.13.
- [VGS02] Natalia Vanetik, Ehud Gudes, and Solomon Eyal Shimony. Computing frequent graph patterns from semistructured data. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 458–465. IEEE, 2002. doi:10.1109/ICDM.2002.1183988.
- [WRF13] Yuyi Wang, Jan Ramon, and Thomas Fannes. An efficiently computable subgraph pattern support measure: counting independent observations. *Data Mining and Knowledge Discovery*, 27(3):444–477, 2013. doi:10.1007/s10618-013-0318-x.
- [Yan81] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7, VLDB '81*, pages 82–94. VLDB Endowment, 1981. doi:10.5555/1286831.1286840.

APPENDIX A. PROOFS OF SECTION 6.2

In this section, we provide missing proofs of some easy lemmas stated in Section 6.2.

A.1. Proof of Lemma 6.2. If $\alpha_1 \neq \alpha_2 \in A|_{X'}$, then there exists $x' \in X'$ such that $\alpha_1(x') \neq \alpha_2(x')$, so if $\gamma_1 \in A[\alpha_1]$ and $\gamma_2 \in A[\alpha_2]$ then $\gamma_1(x') = \alpha_1(x') \neq \alpha_2(x') = \gamma_2(x')$.

A.2. Proof of Lemma 6.3. First note that the union on the right is disjoint by Lemma 6.2.

For the inclusion from the left to the right, let $\alpha \in A[\alpha'']$ and $\alpha' = \alpha|_{X'}$. By definition, $\alpha' \in A|_{X'}$ so $\alpha \in A[\alpha']$. Furthermore, $\alpha' \in A|_{X'}[\alpha'']$ so $\alpha \in \biguplus_{\tilde{\alpha}' \in A|_{X'}[\alpha'']} A[\tilde{\alpha}']$.

For the inclusion from the right to the left, let $\alpha \in \biguplus_{\alpha' \in A|_{X'}[\alpha'']} A[\alpha']$ and let $\alpha' \in A|_{X'}[\alpha'']$ be such that $\alpha \in A[\alpha']$. By definition, $\alpha|_{X'} = \alpha'$ and $\alpha'|_{X''} = \alpha''$. Since $X'' \subseteq X'$, $\alpha|_{X''} = \alpha'|_{X''} = \alpha''$. Thus $\alpha \in A[\alpha'']$.

A.3. Proof of Lemma 6.4. Let $\alpha'' \in A|_{X''}$. We have:

$$\begin{aligned}
\pi_{X''}(\omega)(\alpha'') &= \sum_{\alpha \in A[\alpha'']} \omega(\alpha) && \text{by definition} \\
&= \sum_{\alpha' \in A|_{X'}[\alpha'']} \sum_{\alpha \in A[\alpha']} \omega(\alpha) && \text{by Lemma 6.3} \\
&= \sum_{\alpha' \in A|_{X'}[\alpha'']} \pi_{X'}(\omega)(\alpha') && \text{by definition of } \pi_{X'}(\omega) \\
&= \pi_{X''}(\pi_{X'}(\omega))(\alpha'') && \text{by definition of } \pi_{X''}(\pi_{X'}(\omega)).
\end{aligned}$$

The last equality is well defined since $\alpha'' \in A|_{X''} = (A|_{X'})|_{X''}$.