Linear Programs with Conjunctive Queries

Florent Capelli
Univ. Lille, Inria, CNRS, UMR 9189 - CRIStAL, F-59000 Lille, France

Nicolas Crosetti
Univ. Lille, Inria, CNRS, UMR 9189 - CRIStAL, F-59000 Lille, France

Joachim Niehren
Univ. Lille, Inria, CNRS, UMR 9189 - CRIStAL, F-59000 Lille, France

Jan Ramon
Univ. Lille, Inria, CNRS, UMR 9189 - CRIStAL, F-59000 Lille, France

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 naive 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.

2012 ACM Subject Classification Theory of computation → Logic and databases

Keywords and phrases Database queries, linear programming, hypergraph decomposition

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.5

Related Version *Full Version*: `https://hal.archives-ouvertes.fr/hal-01981553`

Funding 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 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.

Acknowledgements 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.

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 [5] and for constraint programming in MiniZinc [15]. 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 [12], or when using linear optimization to explain the result of a database query to the user as proposed by Meliou and Suciu [14]. Moreover, tools like `SolveDB` [19] 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 $LP(CQ)$ in the present paper. The objective is to become able to specify weightings of answer sets of database queries, that optimize a linear objective function subject to linear constraints. The optimal weightings of $LP(CQ)$ programs can be computed in a naive manner, by first answering the database queries, and then solving a linear program parametrized by the answer sets. 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 the fragment of so-called projecting $LP(CQ)$ programs for which we also bound the hypertree width of the queries. The particular case of width 1 covers the class of acyclic conjunctive queries. By using hypertree decompositions, our algorithm is based on a factorized interpretation of a projecting $LP(CQ)$ program over a database. The factorized interpretation uses other linear program variables, that represent sums of the linear program variables in the naive interpretation. The number of linear program variables in the factorized interpretation depends only on the widths of the hypertree decompositions of the queries in the $LP(CQ)$ program, 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. With respect to the combined complexity, the special case of projecting $LP(CQ)$ programs with bounded quantifier depth becomes tractable for acyclic conjunctive queries, while it is NP-complete in general.

We prove the correctness of the factorized interpretation with respect to the naive interpretation. For this we have to show a correspondence between weightings of answer sets on the naive 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 the tree decomposition.

Conjunctive queries with existential quantifiers are dealt with by showing that one can find an equivalent projecting $LP(CQ)$ program with quantifier free conjunctive queries only.

1.1 Applications

A wide range of applications of linear programs can benefit from conjunctive queries.

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 $D = \mathsf{F} \uplus \mathsf{O} \uplus \mathsf{W} \uplus \mathsf{B} \uplus \mathbb{R}_+$. The elements $d \in D$ encoding a positive real number can be decoded back by applying the database's functions $\mathbf{num}^{\mathbb{D}}$, yielding the positive real number $\mathbf{num}^{\mathbb{D}}(d) \in \mathbb{R}_+$. The database $\mathbb{D}$ has four tables. The first table $prod^{\mathbb{D}} \subseteq \mathsf{F} \times \mathsf{O} \times \mathbb{R}_+$ contains triples (f, o, q) stating that the factory f can produce up to q units of object o. The second table $order^{\mathbb{D}} : \mathsf{B} \times \mathsf{O} \times \mathbb{R}_+$ contains triples (b, o, q) stating that the buyer b orders q units of object o. The third table $store^{\mathbb{D}} \subseteq \mathsf{W} \times \mathbb{R}_+$ contains pairs (w, l) stating that the warehouse w has a storing limit of l.

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

Figure 1 A $LP(CQ)$ program for the resource delivery optimization where $Q = dlr(f', w', b', o')$.

The fourth table $route^{\mathbb{D}} : (\mathsf{F} \times \mathsf{W} \times \mathbb{R}_+) \cup (\mathsf{W} \times \mathsf{B} \times \mathbb{R}_+)$ 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 $Q = dlr(f', w', b', o')$. The goal is to determine for each of these possible deliveries the quantity of the object that should actually be sent. These quantities are modelled by the unknown weights θ^{α}_{Q} of the query answers $\alpha \in sol^{\mathbb{D}}(Q)$. For any factory f and warehouse w the sum $\sum_{\alpha\in sol^{\mathbb{D}}(Q\wedge w'\doteq w\wedge f'\doteq f)} \theta^{\alpha}_{Q}$ is described by the expression $\textbf{weight}_{(f',w',b',o'):f'\doteq f\wedge w'\doteq w}(Q)$ when interpreted over $\mathbb{D}$.

We use the $LP(CQ)$ program in Figure 1 to describe the optimal weights that minimize the overall delivery costs. 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.
- 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^{α}_{Q} for each of the query answers α, we can interpret the $LP(CQ)$ program in Figure 1 as a linear program. However the number of answers of Q and thus the number of variables in this program could be cubic in the size of the database, which quickly grows too big.

Our factorized interpretation for the projecting $LP(CQ)$ program in Figure 1 produces a linear program that only has a quadratic number of variables since the query Q (as well as the whole $LP(CQ)$ program) has a hypertree decomposition of width 2.

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 [4] 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 $LP(CQ)$

with a single conjunctive query that is acyclic, i.e., of hypertree with 1. While the naive 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. The example is worked out in the full version [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. [20] 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 $LP(CQ)$ 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. More information on the $LP(CQ)$ program can be found in the full version [2].

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 [21] 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 [6, 7, 8] 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 [18] or enumerating it with small delay [1]. These tractability results can be obtained in a unified and generalized way by using factorized databases introduced by Olteanu and Závodný [16, 17], 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 [17] 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 databases 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 [1] 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 [17], we choose to directly work on tree decompositions because we need to incorporate some structure of the linear program into our tree decomposition to efficiently handle complex linear programs.

Linear expressions	$S, S' \in L_{\text{E}}$	$::=$	$c \mid \xi \mid cS \mid S + S'$
Linear constraints	$C, C' \in L_{\text{C}}$	$::=$	$S \leq S' \mid C \wedge C' \mid \mathit{true}$
Linear programs	$L \in L_{\text{P}}$	$::=$	**maximize** S **subject to** C

Figure 2 The set of linear programs L_{P} with variables $\xi \in \Xi$ and constants $c \in \mathbb{R}$.

Organization of the paper. Section 2 contains the necessary definitions to understand the paper. Section 3 presents the language $LP(CQ)$ of linear programs parametrized by conjunctive queries and gives its semantics. Section 4 defines a fragment of $LP(CQ)$ for which we propose a more efficient algorithm. Finally, Section 5 presents encouraging practical results on solving the delivery optimization problem using this algorithm. Due to space limit, most proofs and full details on applications to differential privacy and s-measure computation can be found in the full version [2].

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 $f : S \to \mathbb{R}_+$.

Given a set of (total) functions $A \subseteq D^S = \{f \mid f : S \to D\}$ 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 $\alpha : X \to \mathsf{D}$ that maps elements of a finite subset of variables $X \subseteq \mathcal{X}$ to values of D. For any two sets of variable assignments $A_1 \subseteq \mathsf{D}^{X_1}$ and $A_2 \subseteq \mathsf{D}^{X_2}$ 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 $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$ we denote by $\mathit{set}(\mathbf{x}) = \{x_1, \ldots, x_n\}$ the set of the elements of $\mathbf{x}$. For any variable assignment $\alpha : X \to D$ with $\mathit{set}(\mathbf{x}) \subseteq X$ we denote the application of the assignment α on $\mathbf{x}$ by $\alpha(\mathbf{x}) = (\alpha(x_1), \ldots, \alpha(x_n))$.

Linear programs. Let Ξ be a set of linear program variables. In Figure 2, we recall the definition of the sets of linear expressions L_{E}, linear constraints L_{C}, and linear programs L_{P} with variables in Ξ. We consider the usual linear equations $S \doteq S'$ as syntactic sugar for the constraints $S \leq S' \wedge S' \leq S$. For any linear program $L =$ **maximize** S **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 **maximize** $-1\ S$ **subject to** C up to negation.

The formal semantics of linear programs is recalled in the full version [2]. Since we will only be interested in variables for positive real numbers – and do not want to impose positivity constraints all over – we restrict variables of linear programs to always be positive

$$\begin{array}{lll} \text{Expressions} & E_1, \ldots, E_n \in Ex_{\mathcal{C}} & ::= x \mid a \\ \text{Conjunctive queries} & Q, Q' \in C\mathrm{Q}_\Sigma & ::= E_1 \doteq E_2 \mid r(E_1, \ldots, E_n) \mid Q \wedge Q' \mid \exists x.Q \mid \mathit{true} \end{array}$$

Figure 3 The set of conjunctive queries $C\mathrm{Q}_\Sigma$ with schema $\Sigma = ((\mathcal{R}^{(n)})_{n\in\mathbb{N}}, \mathcal{C})$ where $x \in \mathcal{X}$, $a \in \mathcal{C}$, and $r \in \mathcal{R}^{(n)}$.

real numbers. For any weightings $\omega : \Xi \to \mathbb{R}_+$, the value of a sum $S \in L\mathrm{E}$ is the real number $[\![S]\!]_\omega \in \mathbb{R}$, and the value of a constraint $C \in L\mathrm{C}$ is the truth value $[\![C]\!]_\omega \in \mathbb{B}$. The optimal solution $[\![L]\!] \in \mathbb{R}$ of a linear program L with objective function S and constraint C is $[\![L]\!] = \max\{[\![S]\!]_\omega \mid \omega : \Xi \to \mathbb{R}_+,\ [\![C]\!]_\omega = 1\}$. It is well-known that the optimal solution of a linear program can be computed in polynomial time [10].

Rooted trees. A digraph is a pair $(\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$ and edge sets $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. 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 by $\uparrow u = (\mathcal{V} \setminus \downarrow u) \cup \{u\}$ the context of u. A *rooted tree* is an acyclic digraph where $(u, v), (u', v) \in \mathcal{E}$ implies $u = u'$, and 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. Observe that in this tree, the paths are oriented from the root to the leaves of the tree.

Relational Databases. A *database schema* is a pair $\Sigma = (R, \mathcal{C})$ where $\mathcal{C}$ a finite set of constants ranged over by a, b and $R = \cup_{n\in\mathbb{N}}\mathcal{R}^{(n)}$ is a finite set of relation symbols. The elements $r \in \mathcal{R}^{(n)}$ are called relation symbols of arity $n \in \mathbb{N}$.

A *database* $\mathbb{D} \in db_\Sigma$ is a tuple $\mathbb{D} = (\Sigma, D, \cdot^{\mathbb{D}})$, where Σ is a schema, D a finite set of database elements, and $r^{\mathbb{D}} \subseteq D^n$ a relation for any relation symbol $r \in \mathcal{R}^{(n)}$ and $a^{\mathbb{D}} \in D$ a database element for any constant $a \in \mathcal{C}$. We also define the database's domain $\mathit{dom}(\mathbb{D}) = D$.

A *database with real numbers* is a tuple $\mathbb{D} = (\Sigma, D, \cdot^{\mathbb{D}}, \mathbf{num}^{\mathbb{D}})$ such that $\mathbb{D} = (\Sigma, D, \cdot^{\mathbb{D}})$ is a relational database and $\mathbf{num}^{\mathbb{D}}$ a partial function from D to $\mathbb{R}$.

Conjunctive Queries. In Figure 3 we recall the notion of conjunctive queries on relational databases. An expression $E \in Ex_{\mathcal{C}}$ is either a (query) variable $x \in \mathcal{X}$ or a constant $a \in \mathcal{C}$. The set of conjunctive queries $Q \in C\mathrm{Q}_\Sigma$ is built from equations $E_1 \doteq E_2$, atoms $r(E_1, \ldots, E_n)$, the logical operators of conjunction $Q \wedge Q'$ and existential quantification $\exists x.Q$. Given a vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$ and a query Q, we write $\exists \mathbf{x}.Q$ instead of $\exists x_1. \ldots . \exists x_n.Q$.

The set of free variables $\mathit{fv}(Q) \subseteq \mathcal{X}$ are those variables that occur in Q outside the scope of an existential quantifier. A conjunctive query Q is said to be *quantifier free* if it does not contain any existential quantifier.

For any conjunctive query $Q \in C\mathrm{Q}_\Sigma$, set $X \supseteq \mathit{fv}(Q)$ and database $\mathbb{D} \in db_\Sigma$ we define the answer set $\mathit{sol}^{\mathbb{D}}_X(Q)$. It contains all those assignments $\alpha : X \to \mathit{dom}(\mathbb{D})$ for which Q becomes true on $\mathbb{D}$. We also write $\mathit{sol}^{\mathbb{D}}(Q)$ instead of $\mathit{sol}^{\mathbb{D}}_{\mathit{fv}(Q)}(Q)$. Observe that $\mathit{sol}^{\mathbb{D}}(\exists \mathbf{x}.Q) = \mathit{sol}^{\mathbb{D}}(Q)_{|\mathit{fv}(Q)\setminus \mathit{set}(\mathbf{x})}$.

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.

▶ **Definition 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 $\mathcal{B} : \mathcal{V} \to 2^X$ 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** (Hypertree width of quantifier free conjunctive queries)**.** *Let $Q \in C_{Q_\Sigma}$ 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.*

We call a conjunctive query α-acyclic if it has general hypertree width 1. The query $r(x, y) \wedge r(y, z)$ has the generalized hypertree decomposition $(\mathcal{V}, \mathcal{E}, \mathcal{B})$ with $\mathcal{V} = \{1, 2, 3\}$, $\mathcal{E} = \{(1, 2), (1, 3)\}$, and $\mathcal{B} = [1/\{y\}, 2/\{x, y\}, 3/\{y, z\}]$ of width 1, so it is α-acyclic.

Many problems can be solved efficiently on conjunctive queries having a small hypertree width. We will mainly be interested in the problem of efficiently computing $sol^{\mathbb{D}}(Q)$.

▶ **Lemma 3.** *Given a tree decomposition $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ of a quantifier free conjunctive query $Q \in C_{Q_\Sigma}$ of width k and a database $\mathbb{D} \in db_\Sigma$, one can compute the collection of bag projections $(sol^{\mathbb{D}}(Q)_{|\mathcal{B}(u)})_{u \in \mathcal{V}}$ in time $O((|\mathbb{D}|^k \log(|\mathbb{D}|)) \cdot |T|)$.*

Lemma 3 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 [13, Theorem 6.25]. This yields a superset S_u of $sol^{\mathbb{D}}(Q)|_{\mathcal{B}(u)}$ 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 $sol^{\mathbb{D}}(Q)$.

Note that if Q contains n atoms, $sol^{\mathbb{D}}(Q)$ may be of size $O(|\mathbb{D}|^n)$ while $(sol^{\mathbb{D}}(Q)_{|\mathcal{B}(u)})_{u \in \mathcal{V}}$ has size $O(|\mathbb{D}|^k \cdot |T|)$ where k is the width of T. In the particular case of α-acyclic conjunctive queries, where $k = 1$, the overall size of the projections is linear. It gives a succinct way of describing the set of solutions of Q that we exploit in this paper.

We observe here that Lemma 3 is still valid if one replaces hypertree width with the more general notion of fractional hypertree width [9]. Since our tractability results only follows from the bounds presented in Lemma 3, it implies that our results also holds for tree decomposition having bounded fractional hypertree width. We however choose to present our result by using hypertree width since it is easier to define.

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

▶ **Definition 4.** *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 $u_1, ..., u_k$ with $\mathcal{B}(u) = \mathcal{B}(u_1) = ... = \mathcal{B}(u_k)$.*

We call T normalized [1] *if all its nodes in $\mathcal{V}$ are either extend nodes, project nodes, join nodes, or leaves.*

[1] In the literature this property is referred to as "nice" tree decompositions.

Constant numbers	$N \in Num_{\mathcal{C}}$	::=	$c \mid \mathbf{num}(E)$
$LP(CQ)$ expressions	$S, S' \in L_{\mathrm{E}\Sigma}$	::=	$\mathbf{weight}_{\mathbf{x}:Q'}(Q) \mid \sum_{\mathbf{x}:Q} S \mid NS \mid S + S' \mid N$
$LP(CQ)$ constraints	$C, C' \in L_{\mathrm{C}\Sigma}$	::=	$S \leq S' \mid C \wedge C' \mid \mathit{true} \mid \forall \mathbf{x}{:}Q.C$
$LP(CQ)$ programs	$L \in L_{\mathrm{P}\Sigma}$	::=	$\mathbf{maximize}\ S\ \mathbf{subject\ to}\ C$
			where $fv(S) = fv(C) = \emptyset$.

Figure 4 The set of $LP(CQ)$ programs $L_{\mathrm{P}\Sigma}$ where $c \in \mathbb{R}$, $E \in Ex_{\mathcal{C}}$, $\mathbf{x} \in \mathcal{X}^*$ and $Q, Q' \in C_{\mathrm{Q}\Sigma}$.

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 5** (Lemma of 13.1.2 of [11]). *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 Conjunctive Queries

We want to assign weights to the answers of a conjunctive query on a database, such that they maximize a linear objective function subject to linear constraints. For this, we introduce the language $LP(CQ)$ of linear programs with conjunctive queries.

3.1 Syntax

Its syntax is given in Figure 4. Note that an example of an $LP(CQ)$ program for optimal warehouse selection was already given in Figure 1. $LP(CQ)$ programs are interpreted as linear programs whose variables describe the solutions of conjunctive queries. As a consequence, they do *not* contain any explicit linear program variables. Instead, they may contain weight expressions $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ over conjunctive queries $Q, Q' \in C_{\mathrm{Q}\Sigma}$. Intuitively, this expression is interpreted as a linear expression over linear program variables representing a solution of $Q \wedge Q'$. Variables of Q and Q' however may be bound in the context, for example through universal quantifiers or Σ-operators. The query variables in $\mathbf{x}$ are bound by the expression taking scope over Q and Q'. The free (query) variable of weight expressions must however be bound by the context, so that they will be instantiated to some database values before evaluation. Weight expressions without free variables reason about an unknown weighting of the answer set of query Q on the given database $\mathbb{D}$ with the variables in $set(\mathbf{x})$. Its value is then the sum over the weights of tuples in the answer set of $Q \wedge Q'$ on the database $\mathbb{D}$ with variables in $set(\mathbf{x})$.

Beside weight expressions, $LP(CQ)$ expressions in $L_{\mathrm{E}\Sigma}$ may also contain expression $N \in Num_{\mathcal{C}}$ or NS where $S \in L_{\mathrm{E}\Sigma}$ and N is a constant number expression, which is either a real number $c \in \mathbb{R}$ or a number expression $\mathbf{num}(E)$ with $E \in \mathcal{X} \cup \mathcal{C}$. An expression $\mathbf{num}(a)$ denotes the real number $\mathbf{num}^{\mathbb{D}}(a^{\mathbb{D}})$ if this value is defined. Note that the real value of $\mathbf{num}(a)$ over $\mathbb{D}$ is constant from the perspective of the linear program once the database $\mathbb{D}$ is fixed. $LP(CQ)$ constrains $C \in L_{\mathrm{C}\Sigma}$ are conjunctions of inequalities $S \leq S'$ between $LP(CQ)$ expressions $S, S' \in L_{\mathrm{E}\Sigma}$, and universally quantified constraints $\forall \mathbf{x}{:}Q.C'$ requiring that C' must be valid for all possible values of $\mathbf{x}$ in the solution of Q over the database (after instantiation of the free variables of $\forall \mathbf{x}{:}Q.C'$). The bound variables in $\mathbf{x}$ have scope over Q and C.

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

Figure 5 Free variables of $LP(CQ)$ expressions, constraints, and programs.

$LP(CQ)$ programs $L \in L\mathrm{P}_\Sigma$ are built from $LP(CQ)$ expressions and $LP(CQ)$ constraints as one might expect. Note, however, that free query variables are ruled out at this level, while being permitted in nested $LP(CQ)$ constraints in $L\mathrm{C}_\Sigma$ and expressions in $L\mathrm{E}_\Sigma$.

The sets of free variables of $LP(CQ)$ expressions, constraints, and programs are formally defined in Figure 5. For instance, the following $LP(CQ)$ constraint C from the warehouse example has three free variables in $fv(C) = \{f, o, q\}$:

$$\mathbf{weight}_{(f',w',b',o'):f' \doteq f \wedge o' \doteq o}(dlr(f', w', b', o')) \leq \mathbf{num}(q)$$

The variables f', w', b', o' are bound by the weight expression. The free variables f, o, q are bound by a quantifier in the context, which in the resource delivery example is the universal quantifier $\forall(f, o, q){:}prod(f, o, q)$.

3.2 Semantics

We next define the semantics of an $LP(CQ)$ program $L \in L\mathrm{P}_\Sigma$ with respect to a database $\mathbb{D} \in db_\Sigma$ with real numbers by an interpretation to a linear program $\langle L \rangle^{\mathbb{D}} \in L\mathrm{P}$, that we will refer to as the naive interpretation from now on.

For doing so, one step is to replace the free variables of the queries of $LP(CQ)$ programs by elements from the database. For this we assume that we have constants for all elements of the database domain, that is $dom(\mathbb{D}) \subseteq \mathcal{C}$. We then define for any conjunctive query Q and variable assignment $\gamma : Y \to D$ a conjunctive query $sbs_\gamma(Q)$, by replacing in Q all free occurrences of variables $y \in Y$ in Q by $\gamma(y)$. The formal definition is given in the full version [2].

In order to define the semantics of an $LP(CQ)$ program L over a database $\mathbb{D}$ we consider the following set of linear program variables:

$$\Theta_L^{\mathbb{D}} = \{\theta^{\alpha}_{sbs_\gamma(Q)} \mid S = \mathbf{weight}_{\mathbf{x}:Q'}(Q) \text{ in } L, \alpha : set(\mathbf{x}) \to dom(\mathbb{D}),\ \gamma : fv(S) \to dom(\mathbb{D})\}$$

Let $S = \mathbf{weight}_{\mathbf{x}:Q'}(Q)$ be a weight expression and $\gamma : Y \to dom(\mathbb{D})$ a variable assignment for the free variables $fv(S) \subseteq Y$ such that $set(\mathbf{x}) \cap Y = \emptyset$. The interpretation of the weight expression $\langle S \rangle^{\mathbb{D},\gamma}$ is the overall weight of the solutions $\alpha \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q' \wedge Q))$ where $\tilde{\gamma} = \gamma_{|Y \setminus set(\mathbf{x})}$ in the table $sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q))$. It is described by the following linear expression:

$$\langle S \rangle^{\mathbb{D},\gamma} = \sum_{\alpha \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q \wedge Q'))} \theta^{\alpha}_{sbs_{\tilde{\gamma}}(Q)}$$

The (naive) interpretations $\langle S \rangle^{\mathbb{D},\gamma}$ and $\langle C \rangle^{\mathbb{D},\gamma}$ of other kinds of $LP(CQ)$ expressions $S \in L\mathrm{E}_\Sigma$ and constraints $C \in L\mathrm{C}_\Sigma$ over a database $\mathbb{D}$ and an environment γ are rather obvious. Note that $LP(CQ)$ programs L can be interpreted as linear programs $\langle L \rangle^{\mathbb{D}} \in L\mathrm{P}$ without any environment as they do not have free variables. The definitions are summarized in Figure 6.

$$\langle \mathbf{weight}_{\mathbf{x}:Q'}(Q)\rangle^{\mathbb{D},\gamma} = \sum_{\alpha \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q \wedge Q'))} \theta^{\alpha}_{sbs_{\tilde{\gamma}}(Q)}$$
$$\langle \forall \mathbf{x}{:}Q.C\rangle^{\mathbb{D},\gamma} = \bigwedge_{\gamma' \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q))} \langle C\rangle^{\mathbb{D},\tilde{\gamma}\cup\gamma'}$$
$$\langle \textstyle\sum_{\mathbf{x}:Q} S\rangle^{\mathbb{D},\gamma} = \sum_{\gamma' \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q))} \langle S\rangle^{\mathbb{D},\tilde{\gamma}\cup\gamma'}$$
$$\langle NS\rangle^{\mathbb{D},\gamma} = \langle N\rangle^{\mathbb{D},\gamma}\langle S\rangle^{\mathbb{D},\gamma}$$
$$\langle \mathbf{num}(a)\rangle^{\mathbb{D},\gamma} = \mathbf{num}^{\mathbb{D}}(a^{\mathbb{D}}) \qquad \text{(may be undefined)}$$

$$\langle S_1 + S_2\rangle^{\mathbb{D},\gamma} = \langle S_1\rangle^{\mathbb{D},\gamma} + \langle S_2\rangle^{\mathbb{D},\gamma}$$
$$\langle S_1 \leq S_2\rangle^{\mathbb{D},\gamma} = \langle S_1\rangle^{\mathbb{D},\gamma} \leq \langle S_2\rangle^{\mathbb{D},\gamma}$$
$$\langle C_1 \wedge C_2\rangle^{\mathbb{D},\gamma} = \langle C_1\rangle^{\mathbb{D},\gamma} \wedge \langle C_2\rangle^{\mathbb{D},\gamma}$$
$$\langle true\rangle^{\mathbb{D},\gamma} = true$$
$$\langle c\rangle^{\mathbb{D},\gamma} = c$$

$$\langle \mathbf{maximize}\ S\ \mathbf{subject\ to}\ C\rangle^{\mathbb{D}} = \mathbf{maximize}\ \langle S\rangle^{\mathbb{D},\emptyset}\ \mathbf{subject\ to}\ \langle C\rangle^{\mathbb{D},\emptyset}$$

Figure 6 Naïve interpretation of linear expressions (constraints and programs) with conjunctive queries F over database $\mathbb{D}$ as standard linear expression (constraints and programs respectively) $F^{\mathbb{D},\gamma}$, where $\gamma : Y \to dom(\mathbb{D})$ and $fv(F) \subseteq Y \subseteq \mathcal{X}$ and $\tilde{\gamma} = \gamma_{|Y \setminus set(\mathbf{x})}$.

We note that α-renaming the bound variables in weight expressions does *not* always preserve the semantics of $LP(CQ)$ programs. It may make previously equal queries different, so that different weights may be assigned to their answer sets.

3.3 Simple example

We start with the conjunctive query $Q = R(x') \wedge R(y')$ and the database $\mathbb{D}$ with table $R^{\mathbb{D}} = \{(0),(1)\}$. The answer set of Q is $sol^{\mathbb{D}}(Q) = \{\alpha \mid \alpha : \{x', y'\} \to \{0,1\}\}$. We then consider the following $LP(CQ)$ program L:

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

The naive interpretation $\langle L\rangle^{\mathbb{D}}$ is the following linear program with variables in $\Theta^{\mathbb{D}}_{L}$, where we denote any query answer $\alpha \in sol^{\mathbb{D}}(Q)$ by a pair $(\alpha(x'), \alpha(y'))$ in the cartesian product $\{0,1\}^2$ for simplicity:

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

The objective function $\theta_Q^{(0,0)} + \theta_Q^{(0,1)} + \theta_Q^{(1,0)} + \theta_Q^{(1,1)}$ is the naive interpretation of the expression $\mathbf{weight}_{(x',y'):true}(Q)$. The left-hand side of the first constraint $\theta_Q^{(0,0)} + \theta_Q^{(0,1)}$ is obtained by naivly interpreting the expression $\mathbf{weight}_{(x',y'):x' \doteq x}(Q)$ with free variable x in the environment $[x/0]$. Note that these sums share the linear program variables $\theta_Q^{(0,0)}$ and $\theta_Q^{(0,1)}$, so the two weight expressions of L are semantically related.

3.4 Hardness of solving $LP(CQ)$ programs

In this section, we consider study the complexity of the problem $\mathsf{DECIDE}_{\neq 0}(LP(CQ))$ of deciding whether the optimal value of $\langle L\rangle^{\mathbb{D}}$ given an $LP(CQ)$ L and a database $\mathbb{D}$ is non-zero.

▶ **Theorem 6.** $\mathsf{DECIDE}_{\neq 0}(LP(CQ))$ *is NP-hard.*

Proof. The proof follows by reduction to the problem of deciding whether $sol^{\mathbb{D}}(Q) \neq \emptyset$ given a conjunctive query Q and a database $\mathbb{D}$ in the input, which is known to be *NP*-complete [3]. We consider, given a conjunctive query Q, the following $LP(CQ)$: $L_Q :=$ **maximize weight**$_{\mathbf{x}:true}(Q)$ **subject to weight**$_{\mathbf{x}:true}(Q) \leq 1$.

The hardness now follows from the fact that for every Q and $\mathbb{D}$, $\langle L_Q \rangle^{\mathbb{D}} \neq 0$ if and only if $sol^{\mathbb{D}}(Q) \neq \emptyset$. Indeed, if $sol^{\mathbb{D}}(Q) = \emptyset$, then $\langle L_Q \rangle^{\mathbb{D}} =$ **maximize** 0 **subject to** $0 \leq 1$. The optimal value of $\langle L_Q \rangle^{\mathbb{D}}$ is thus 0. However, if $sol^{\mathbb{D}}(Q) \neq \emptyset$, we have $\langle L_Q \rangle^{\mathbb{D}} =$ **maximize** ϕ **subject to** $\phi \leq 1$ where $\phi = \sum_{\alpha \in sol^{\mathbb{D}}(Q)} \theta_Q^\alpha$. Let $\alpha \in sol^{\mathbb{D}}(Q)$ and consider the weighting ω_α such that $\omega_\alpha(\theta_Q^\alpha) := 1$ and for every $\alpha' \in sol^{\mathbb{D}}(Q)$ such that $\alpha' \neq \alpha$, $\omega_\alpha(\theta_Q^{\alpha'}) := 0$. This weighting clearly respects the constraints of $\langle L_Q \rangle^{\mathbb{D}}$ and has value 1, showing that the optimal value of $\langle L_Q \rangle^{\mathbb{D}} \geq 1$. ◀

4 An Efficiently Solvable Fragment

Motivated by Theorem 6, we now look for a tractable fragment of $LP(CQ)$. We introduce a subclass of projecting $LP(CQ)$ programs and define a notion of width of $LP(CQ)$ programs in this fragment through a collection of hypertree decompositions of the queries they contain. We then show one can find the optimal solution of such programs L more efficiently than by explicitly computing the interpretation over a database $\mathbb{D}$ as a linear program $\langle L \rangle^{\mathbb{D}}$. For this we will present an alternative factorized interpretation of L to a linear program having fewer variables, while preserving the optimal solution.

4.1 Projecting $LP(CQ)$ Programs

We start with the definition of projecting $LP(CQ)$ programs, whose main restriction resides on how they can use conjunctive queries.

▶ **Definition 7.** *The fragment $LP(CQ)_{proj}$ is the set of $LP(CQ)$ programs L such that:*

- *for any subexpression $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ of L, we have that $set(\mathbf{x}) = fv(Q)$ and Q' is a projecting query of the form $\mathbf{x}' \doteq \mathbf{y}$ with $set(\mathbf{x}') \subseteq set(\mathbf{x})$ and $set(\mathbf{x}) \cap set(\mathbf{y}) = \emptyset$.*
- *for any sum $\sum_{\mathbf{x}:Q} S$ and any universal quantifier $\forall \mathbf{x}{:}Q.C$ of L, the query Q is of the form $\exists \mathbf{z}.r(\mathbf{y})$ for some relation symbol $r \in \mathcal{R}^{(n)}$, vector $\mathbf{y} \in \mathcal{X}^n$ and vector $\mathbf{z} \in \mathcal{X}^*$ such that $set(\mathbf{x}) \subseteq fv(Q)$.*

We denote by $LP(CQ_{qf})_{proj}$ the subset of $LP(CQ)_{proj}$ where every conjunctive query Q appearing in a weight expression is quantifier free.

Any expression $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ of a projecting $LP(CQ)$ program is restricted to projection in Q'. Furthermore Q may not have any variables that are free in the weight expression. This condition ensures that the interpretation in environment γ of Q does not substitute any variables, that is $sbs_\gamma(Q) = Q$. Thus, it is interpreted as a sum over θ_Q^α variables where α are solutions of Q taking the same value $\gamma(\mathbf{y})$ on variables $\mathbf{x}'$. Our algorithm will exploit this fact by utilizing tree decompositions of Q to interpret $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ of $LP(CQ_{qf})_{proj}$ with one variable instead of $|sol^{\mathbb{D}}(Q \wedge Q')|$ needed in the naive interpretation.

Another restriction of $LP(CQ)_{proj}$ is that universal quantifiers and sums are guarded by a database relation. Our algorithm does not exploit the structure of conjunctive queries in universal quantifiers and sums so we interpret these expressions in the same way as in Figure 6. To avoid a blow up in the number of constraints, we chose to guard these constructions.

Hypertree Width of Projecting $LP(CQ)$ Programs. We next lift the concept of generalized hypertree width from quantifier free conjunctive queries to $LP(CQ_{qf})_{proj}$ programs. The complexity of our algorithm will depend on this.

For any program L in $LP(CQ)_{proj}$, we define the set of queries $cqs(L)$ that are weighted when interpreting L as $cqs(L) = \{Q \mid \mathbf{weight}_{\mathbf{x}:Q'}(Q)$ is a subexpression of $L\}$. Observe that the resource delivery problem L is in $LP(CQ)_{proj}$ with $cqs(L) = \{dlr(f', w', b', o')\}$.

▶ **Definition 8.** *Let L be an $LP(CQ_{qf})_{proj}$ program and $\mathcal{T} = (T_Q)_{Q \in cqs(L)}$ a collection of decomposition trees. We call $\mathcal{T}$ a tree decomposition of L if for any expression $\mathbf{weight}_{\mathbf{x}:\mathbf{x}' \doteq \mathbf{y}}(Q)$ in L, $T_Q = (\mathcal{V}_Q, \mathcal{E}_Q, \mathcal{B}_Q)$ is a tree decomposition of Q and there is a node u of T_Q such that $\mathcal{B}_Q(u) = set(\mathbf{x}')$. We define the width of $\mathcal{T}$ to be the maximal width of T_Q for $Q \in cqs(L)$. The size of $\mathcal{T}$ is defined to be $|\mathcal{T}| = \sum_{Q \in cqs(L)} |\mathcal{V}_Q|$.*

The goal of this section is to show how one can take advantage of such tree decompositions in order to solve an $LP(CQ_{qf})_{proj}$ more efficiently than the naive approach of computing its interpretation. To do so, we will use the tree decomposition to compute a smaller interpretation of the linear program that we call *factorized interpretation* and whose definition is given in Section 4.2. The correctness of our approach relies on the following theorem.

▶ **Theorem 9** (Main). *Let L be a $LP(CQ_{qf})_{proj}$ program, $\mathcal{T}$ a decomposition of L of width k and $\mathbb{D}$ a database. The factorized interpretation $\rho^{\mathcal{T},\mathbb{D}}(L)$ from Figure 7 has $O(|\mathcal{T}||\mathbb{D}|^k)$ variables and the same optimal value as the naïve interpretation $\langle L \rangle^{\mathbb{D}}$.*

Observe that the number of variables of $\langle L \rangle^{\mathbb{D}}$ is roughly the total number of solutions of the conjunctive queries in $cqs(L)$, which may be up to $O(|\mathbb{D}|^n)$, where n is the number of atoms in the conjunctive queries. In Theorem 9, the degree of the polynomial now only depends on the width of the queries, which may be much smaller, resulting in a more succinct linear program that is easier to solve. In the resource optimization example, this allows to go from a cubic number of variables to a quadratic one, but the improvement may be much better on queries having many atoms and small width.

4.2 Factorized Interpretation

The idea of the factorized interpretation is to reduce the number of variables by introducing "group" variables. In the simple example from Section 3.3, we can observe that by grouping together the variables $\theta_Q^{(0,0)}$ and $\theta_Q^{(0,1)}$ into a group variable $\xi_{Q,[x/0]}$ and similarly $\theta_Q^{(1,0)}$ and $\theta_Q^{(1,1)}$ into a group variable $\xi_{Q,[x/1]}$ we can reduce the number of variables from four to two, leading to the following linear program:

$$\begin{array}{rl} \textbf{maximize} & \xi_{Q,[x/0]} + \xi_{Q,[x/1]} \\ \textbf{subject to} & \xi_{Q,[x/0]} \leq 1 \\ \wedge & \xi_{Q,[x/1]} \leq 1 \end{array}$$

Using hypertree decompositions we can develop the grouping idea systematically. We start with a projecting $LP(CQ)$ program L with quantifier free conjunctive queries. Let $\mathcal{T} = (T_Q)_{Q \in cqs(L)}$ be a tree decomposition of L of width k where $T_Q = (\mathcal{V}_Q, \mathcal{E}_Q, \mathcal{B}_Q)$. The set of group variables for the factorized interpretation $\rho^{\mathcal{T},\mathbb{D}}(L)$ is defined by:

$$\Xi_L^{\mathcal{T},\mathbb{D}} = \{\xi_{Q,u,\beta} \mid Q \in cqs(L), u \in \mathcal{V}_Q, \beta \in sol^{\mathbb{D}}(Q)_{|\mathcal{B}_Q(u)}\}.$$

Observe that since T_Q is a tree decomposition of Q of width at most k, $sol^{\mathbb{D}}(Q)_{|\mathcal{B}_Q(u)}$ is of size at most $|\mathbb{D}|^k$. Thus we have at most $|\mathcal{T}||\mathbb{D}|^k$ group variables in $\Xi_L^{\mathcal{T},\mathbb{D}}$.

The *$\mathcal{T}$-factorized interpretation $\rho^{\mathcal{T},\mathbb{D}}(L)$ of L* is defined in Figure 7. Since we are using the group variables from $\Xi_L^{\mathcal{T},\mathbb{D}}$, it uses at most $|\mathcal{T}||\mathbb{D}|^k$ many variables as stated in Theorem 9. It mainly mirrors the naive interpretation of Figure 6 but significantly differs in two places: the first one is the way $\mathbf{weight}_{\mathbf{x}:\mathbf{x}' \doteq \mathbf{y}}(Q)$ is interpreted and the second one is the addition of the *local soundness constraints* $lsc^{\mathcal{T},\mathbb{D}}(Q)$ to the program.

$$\rho^{\mathcal{T},\mathbb{D},\gamma}(\forall \mathbf{x}{:}r(\mathbf{x}).C) = \bigwedge_{\gamma' \in sol^{\mathbb{D}}(r(\mathbf{x}))} \rho^{\mathcal{T},\mathbb{D},\gamma\cup\gamma'}(C) \qquad \rho^{\mathcal{T},\mathbb{D},\gamma}(S_1 + S_2) = \rho^{\mathcal{T},\mathbb{D},\gamma}(S_1) + \rho^{\mathcal{T},\mathbb{D},\gamma}(S_2)$$
$$\rho^{\mathcal{T},\mathbb{D},\gamma}(\sum_{\mathbf{x}:r(\mathbf{x})} S) = \sum_{\gamma' \in sol^{\mathbb{D}}(r(\mathbf{x}))} \rho^{\mathcal{T},\mathbb{D},\gamma\cup\gamma'}(S) \qquad \rho^{\mathcal{T},\mathbb{D},\gamma}(S_1 \leq S_2) = \rho^{\mathcal{T},\mathbb{D},\gamma}(S_1) \leq \rho^{\mathcal{T},\mathbb{D},\gamma}(S_2)$$
$$\rho^{\mathcal{T},\mathbb{D},\gamma}(NS) = \rho^{\mathcal{T},\mathbb{D},\gamma}(N)\rho^{\mathcal{T},\mathbb{D},\gamma}(S) \qquad \rho^{\mathcal{T},\mathbb{D},\gamma}(C_1 \wedge C_2) = \rho^{\mathcal{T},\mathbb{D},\gamma}(C_1) \wedge \rho^{\mathcal{T},\mathbb{D},\gamma}(C_2)$$
$$\rho^{\mathcal{T},\mathbb{D},\gamma}(\mathbf{num}(a)) = \mathbf{num}^{\mathbb{D}}(a^{\mathbb{D}}) \qquad \rho^{\mathcal{T},\mathbb{D},\gamma}(true) = true$$
(may be undefined) $\qquad \rho^{\mathcal{T},\mathbb{D},\gamma}(c) = c$

$$\rho^{\mathcal{T},\mathbb{D},\gamma}(\mathbf{weight}_{\mathbf{x}:\mathbf{x}'\doteq\mathbf{y}}(Q)) = \begin{cases} \xi_{Q,u,\beta} & \text{if } \beta = [\mathbf{x}'/\gamma(\mathbf{y})] \in sol^{\mathbb{D}}(Q)_{|\mathcal{B}_Q(u)} \\ 0 & \text{else} \end{cases}$$

where u is a node of T_Q such that $set(\mathbf{x}') = \mathcal{B}_Q(u)$.

$$\rho^{\mathcal{T},\mathbb{D}}(\mathbf{maximize}\ S\ \mathbf{subject\ to}\ C) = \mathbf{maximize}\ \rho^{\mathcal{T},\mathbb{D},\emptyset}(S)\ \mathbf{subject\ to}\ \rho^{\mathcal{T},\mathbb{D},\emptyset}(C) \wedge \bigwedge_{Q\in cqs(L)} lsc^{\mathcal{T},\mathbb{D}}(Q)$$

Local soundness constraints for $Q \in cqs(L)$ where $A = sol^{\mathbb{D}}(Q)$:

$$lsc^{\mathcal{T},\mathbb{D}}(Q) = \bigwedge_{(u,v)\in\mathcal{E}_Q} \bigwedge_{\gamma\in A_{|\mathcal{B}_Q(u)\cap\mathcal{B}_Q(v)}} \sum_{\beta\in A_{|\mathcal{B}_Q(u)},\beta_{|\mathcal{B}_Q(u)}=\gamma} \xi_{Q,u,\beta} \doteq \sum_{\beta'\in A_{|\mathcal{B}_Q(v)},\beta'_{|\mathcal{B}_Q(v)}=\gamma} \xi_{Q,v,\beta'}$$

Figure 7 $\mathcal{T}$-factorized interpretation of $LP(CQ_{qf})_{proj}$ programs with respect to database $\mathbb{D}$.

In the simple example from Section 3.3 we consider the generalized hypertree decomposition $(\mathcal{V},\mathcal{E},\mathcal{B})$ of Q with $\mathcal{V} = \{r,u,v\}$, $\mathcal{E} = \{(r,u),(r,v)\}$, and $\mathcal{B} = [r/\{\}, u/\{x\}, v/\{y\}]$. The factorized interpretation then yields the following linear program, whose form is similar to the one we showed at the beginning of the subsection:

$$\begin{array}{lll} \mathbf{maximize} & \xi_{Q,r,[]} & \\ \mathbf{subject\ to} & \xi_{Q,u,[x/0]} \leq 1 & \\ \wedge & \xi_{Q,u,[x/1]} \leq 1 & \\ \wedge & \xi_{Q,r,[]} = \xi_{Q,u,[x/0]} + \xi_{Q,u,[x/1]} & \text{local soundness constraints } lsc^{\mathcal{T},\mathbb{D}}(Q) \\ \wedge & \xi_{Q,r,[]} = \xi_{Q,v,[y/0]} + \xi_{Q,v,[y/1]} & \end{array}$$

4.3 Complexity

In this section, we analyse the complexity of constructing the factorized interpretation by applying the inductive definition of Figure 7. Let L be an $LP(CQ)_{proj}$ program and $\mathcal{T} = (T_Q)_{Q\in cqs(L)}$ be a tree decomposition of L of width k where $T_Q = (\mathcal{V}_Q,\mathcal{E}_Q,\mathcal{B}_Q)$.. We first start by bounding the number of variables in $\rho^{\mathcal{T},\mathbb{D}}(L)$. The set of variables of $\rho^{\mathcal{T},\mathbb{D}}(L)$ is, by definition, $\Xi_L^{\mathcal{T},\mathbb{D}}$. Variables are indexed by a query Q of $cqs(L)$, a node of u in the tree decomposition T_Q of Q and a partial assignment β of $\mathcal{B}(Q)$, projected on a bag of T_Q. By Lemma 3, we have at most $|\mathbb{D}|^k$ elements in $sol^{\mathbb{D}}(Q)_{|\mathcal{B}_Q(u)}$. Thus, $\Xi_L^{\mathcal{T},\mathbb{D}}$ has size at most $|\mathcal{T}||\mathbb{D}|^k$ and this set of variables can be computed in time $O(|\mathcal{T}||\mathbb{D}|^k \log(|\mathbb{D}|))$ by Lemma 3.

We now evaluate the number of constraints in $\rho^{\mathcal{T},\mathbb{D}}(L)$. It is clear from Figure 7 that $\rho^{\mathcal{T},\mathbb{D}}(L)$ contains the constraints from L where $\forall$ quantifiers have been unfolded plus the local soundness constraints. By definition, there are at most t $|\mathcal{T}||\mathbb{D}|^k$ local soundness constraints. Moreover, once the projection of $sol^{\mathbb{D}}(Q)$ on the bags of T_Q have been precomputed thanks to Lemma 3, it is easy to see that each local soundness constraints can be computed in time $O(|\mathcal{T}||\mathbb{D}|^k \log(|\mathbb{D}|))$ by taking the join of every bags that are neighbor in one tree decomposition.

Now, let $d_\forall$ the the maximum number of nested $\forall$ appearing in L. It is easy to see that $\rho^{\mathcal{T},\mathbb{D}}(\forall \mathbf{x}{:}r(\mathbf{x}).C)$ multiplies the number of constraints in $\rho^{\mathcal{T},\mathbb{D}}(C)$ by $|r| \leq |\mathbb{D}|$. Thus, apart from the local soundness constraints, there are at most $|L||\mathbb{D}|^{d_\forall}$ constraints in $\rho^{\mathcal{T},\mathbb{D}}(L)$.

Finally, we turn to the complexity of constructing these constraints of $\rho^{\mathcal{T},\mathbb{D}}(L)$. To evaluate the constraints of $\rho^{\mathcal{T},\mathbb{D}}(L)$, one has to unfold $\sum_{\mathbf{x}:r(\mathbf{x})} S$ statements. Let d_Σ be the maximum number of nested $\sum$ in L. Each unfolding of a $\sum_{\mathbf{x}:r(\mathbf{x})} S$ leads to at most $|r| \leq |\mathbb{D}|$ inductive construction. Thus, in a quantifier free constraint from L, one has to evaluate at most $|\mathbb{D}|^{d_\Sigma}$ S-terms. The main complexity of evaluating such term comes from computing $\rho^{\mathcal{T},\mathbb{D}}(\mathbf{weight}_{\mathbf{x}:\mathbf{x}'\doteq\mathbf{y}}(Q))$. Each of them can easily be computed by scanning the structure given by Lemma 3 in time $O(|\mathcal{T}||\mathbb{D}|^k)$.

Overall, the complexity of constructing $\rho^{\mathcal{T},\mathbb{D}}(L)$ is thus $O(|\mathcal{T}||\mathbb{D}|^k(\log|\mathbb{D}|+|\mathbb{D}|^k+|\mathbb{D}|^{d_\forall+d_\Sigma})$.

4.4 Correctness

We now discuss how to prove the correctness statement of Theorem 9 that $\rho^{\mathcal{T},\mathbb{D}}(L)$ has the same optimal value as $\langle L\rangle^{\mathbb{D}}$.

One can see that given a context γ such that $\rho^{\mathcal{T},\mathbb{D},\gamma}(\mathbf{weight}_{\mathbf{x}:\mathbf{x}'\doteq\mathbf{y}}(Q)) = \xi_{Q,u,\beta}$, the usual interpretation would have been $\langle\mathbf{weight}_{\mathbf{x}:\mathbf{x}'\doteq\mathbf{y}}(Q)\rangle^{\mathbb{D},\gamma} = \sum_{\alpha\in sol^{\mathbb{D}}_{set(\mathbf{x})}(Q):\alpha|_{\mathbf{x}'}=\beta}\theta^\alpha_Q$, that is, intuitively, $\xi_{Q,u,\beta}$ represents the linear sum of variables θ^α_Q in the naive interpretation with α compatible with β.

For general programs L we will reconstruct a solution to $\langle L\rangle^{\mathbb{D}}$ from a solution to $\rho^{\mathcal{T},\mathbb{D}}(L)$ such that the value of $\xi_{Q,u,\beta}$ indeed corresponds to the sum of the values of variables θ^α_Q with α compatible with β and vice-versa. To ensure that this is always possible, we have to be careful that variables $\xi_{Q,u,\beta}$ and $\xi_{Q,v,\beta'}$ are compatible with one another because they may correspond to two sums on θ^α_Q variables having a non-empty intersection. We ensure this through *local soundness constraints* $lsc^{\mathcal{T},\mathbb{D}}(Q)$ for every query $Q \in cqs(L)$.

Weightings on Tree Decompositions. One can observe that the key idea in the definition of $\rho^{\mathcal{T},\mathbb{D}}(L)$ is to introduce linear program variables that will intuitively encode the sum of several linear program variables in the naive interpretation $\langle L\rangle^{\mathbb{D}}$. A solution to $\langle L\rangle^{\mathbb{D}}$ maps a variable θ^α_Q to a non-negative real number where $\alpha \in sol^{\mathbb{D}}(Q)$. In other words, it assigns a weight $\omega(\alpha) \in \mathbb{R}_+$ to every $\alpha \in sol^{\mathbb{D}}(Q)$ for every $Q \in cqs(L)$. A solution to $\rho^{\mathcal{T},\mathbb{D}}(L)$ maps a variable $\xi_{Q,u,\beta}$ to a non-negative real number where $\beta \in sol^{\mathbb{D}}(Q)_{|\mathcal{B}_Q(u)}$. In other words, it assigns a weight W_u to every β that is in $sol^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$ for every node u of T_Q.

To reconstruct a solution of $\langle L\rangle^{\mathbb{D}}$ from a solution W of $\rho^{\mathcal{T},\mathbb{D}}(L)$, we need to be able to reconstruct a weighting ω of $sol^{\mathbb{D}}(Q)$ such that $\sum_{\alpha|_{\mathcal{B}_Q(u)}=\beta}\omega(\alpha) = W_u(\beta)$. In this section, we explain that this is always possible as long as the W_u are compatible with one another, which is ensured by local soundness constraints $lsc^{\mathcal{T},\mathbb{D}}(Q)$ in $\rho^{\mathcal{T},\mathbb{D}}(L)$.

The technique is not specifically tied to the fact that the weights are assigned to the solutions of a quantifier free conjunctive query, thus we formulate our result in a more general setting by considering weightings on a set $A \subseteq D^X = \{\alpha \mid \alpha : X \to D\}$ for a finite set of variables X. Intuitively however, one can think of A as $sol^{\mathbb{D}}(Q)$ for **a quantifier-free conjunctive query** Q.

We start by introducing a few notations. Let $X' \subseteq X \subseteq \mathcal{X}$. For any $\alpha' : X' \to D$ we define the set of its extensions into A by $A[\alpha'] = \{\alpha \in A \mid \alpha_{|X'} = \alpha'\}$. Moreover, given a weighting $\omega : A \to \mathbb{R}_+$ of A, we define the projection $\pi_{X'}(\omega) : A_{|X'} \to \mathbb{R}_+$ such that for all $\alpha' \in A_{|X'}$: $\pi_{X'}(\omega)(\alpha') = \sum_{\alpha\in A[\alpha']}\omega(\alpha)$.

We now fix $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ a decomposition tree for X. Given two nodes $u, v \in \mathcal{V}$ we denote the intersection of their bags by $\mathcal{B}^{uv} = \mathcal{B}(u) \cap \mathcal{B}(v)$.

▶ **Definition 10.** *A family $W = (W_v)_{v \in \mathcal{V}}$ is a* weighting collection on T for A *if it satisfies the following conditions for any two nodes $u, v \in \mathcal{V}$:*
- **W_u is a weighting of $A_{|\mathcal{B}(u)}$**, *i.e.,* $W_u : A_{|\mathcal{B}(u)} \to \mathbb{R}_+$.
- **W_u is sound for T at $\{u, v\}$**, *i.e.,* $\pi_{\mathcal{B}^{uv}}(W_u) = \pi_{\mathcal{B}^{uv}}(W_v)$.

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

▶ **Proposition 11.** *For any weighting $\omega : A \to \mathbb{R}_+$, the family $(\pi_{\mathcal{B}(v)}(\omega))_{v \in \mathcal{V}}$ is a weighting collection on T for A.*

What is more interesting is the other way around, that is, given $(W_u)_{u \in \mathcal{V}}$ a weighting collection on T, whether we can find a weighting ω of A such that $W_u = \pi_{\mathcal{B}(u)}(\omega)$ for every u. It turns out that soundness is not enough to ensure the existence of such a weighting. However it becomes possible when A is conjunctively decomposed:

▶ **Definition 12.** *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 $\beta \in A_{|\mathcal{B}(u)}$: $\{\alpha_1 \cup \alpha_2 \mid \alpha_1 \in A_{|\mathcal{B}(\uparrow u)}[\beta],\ \alpha_2 \in A_{|\mathcal{B}(\downarrow u)}[\beta]\} \subseteq A[\beta]$ where $\mathcal{B}(V) = \bigcup_{v \in V} \mathcal{B}(v)$ for any $V \subseteq \mathcal{V}$.*

Note that the inverse inclusion holds in general. Of course, this property holds if A is the answer set of a conjunctive queries and the tree is a tree decomposition of Q:

▶ **Proposition 13.** *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 $sol^{\mathbb{D}}(Q)$ is conjunctively decomposed by T.*

Proposition 13 does not hold when Q is not quantifier free. It explains why the technique only works for the fragment $LP(CQ_{qf})_{proj}$. We however explain how one can use the same technique on $LP(CQ)_{proj}$ in Section 4.5.

Soundness and conjunctive decomposition are enough to prove this correspondence theorem that allows us to transform solutions of $\rho^{\mathcal{T},\mathbb{D}}(L)$ to solutions of $\langle L \rangle^{\mathbb{D}}$ and vice-versa.

▶ **Theorem 14** (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.*
1. *For every weighting ω of A, $(\pi_{\mathcal{B}(u)}(\omega))_{u \in \mathcal{V}}$ is a weighting collection on T for A.*
2. *For any weighting collection W on T for A there exists a weighting ω of A such that $\forall u : W_u = \pi_{\mathcal{B}(u)}(\omega)$.*

While the first item of Theorem 14 follows by Proposition 11 and can be proven by a simple calculation, the second item is harder to prove. We present here one way of constructing ω from $(W_u)_{u \in \mathcal{V}}$. The proof of correctness of this construction can be found in the full version [2].

Let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a normalized decomposition tree of X and $W = (W_u)_{u \in \mathcal{V}}$ a weighting collection on T for $A \subseteq D^X$. For any node $u \in \mathcal{V}$, we inductively construct $\omega_u : A_{|\mathcal{B}(\downarrow u)} \to \mathbb{R}_+$.

If u is a leaf of T, we define ω_u such that for all $\alpha \in A_{|\mathcal{B}(\downarrow u)}$, $\omega_u(\alpha) := W_u(\alpha)$.

Now, assume $\omega_{u'}$ is defined for all children u' of u. Let $\alpha \in A_{|\mathcal{B}(\downarrow u)}$ and denote by $\beta = \alpha_{|\mathcal{B}(u)}$. We define $\omega_u(\alpha)$ as follows:

If u is an extend node with a single child v then $\omega_u(\alpha) := \frac{W_u(\beta)}{W_v(\alpha_{|\mathcal{B}(v)})}\omega_v(\alpha_{|\mathcal{B}(\downarrow v)})$ if $W_v(\alpha_{|\mathcal{B}(v)}) > 0$ and $\omega_u(\alpha) := 0$ otherwise.

If u is a project node with a single child v then $\omega_u(\alpha) := \omega_v(\alpha_{|\mathcal{B}(\downarrow v)})$.

If u is a join node with children $v_1, \ldots, v_k$ then $\omega_u(\alpha) := \frac{\prod_{i=1}^{k} \omega_{v_i}(\alpha_{|\mathcal{B}(\downarrow v_i)})}{W_u(\beta)^{k-1}}$ if $W_u(\beta) > 0$ and $\omega_u(\alpha) := 0$ otherwise.

Finally, we let ω be ω_r where r is the root of T. The proof that $\forall u : W_u = \pi_{\mathcal{B}(u)}(\omega)$ is done via two inductions. The first one is a bottom-up induction to prove that $W_u = \pi_{\mathcal{B}(u)}(\omega)$ for every node u in the tree decomposition. Then, by top-down induction, one can prove that $\omega_u = \pi_{\mathcal{B}(\downarrow u)}(\omega_r)$. The proof is tedious and mainly relies on calculations and careful analysis on how A is decomposed along T.

Correctness Proof. We are now ready to prove that, given a tree decomposition $\mathcal{T}$ of a $LP(CQ)$ program L of $LP(CQ_{qf})_{proj}$, $\rho^{\mathcal{T},\mathbb{D}}(L)$ and $\langle L \rangle^{\mathbb{D}}$ have the same optimal value.

For any weighting $\dot{\omega}$: $\Theta_L \to \mathbb{R}_+$ we define a weighting $\Pi(\dot{\omega}) : \Xi_L^{\mathcal{T},\mathbb{D}} \to \mathbb{R}_+$ such that for all $\xi_{Q,u,\beta} \in \Xi_L^{\mathcal{T},\mathbb{D}}$: $\Pi(\dot{\omega})(\xi_{Q,u,\beta}) = \sum_{\alpha \in sol^{\mathbb{D}}(Q)[\beta]} \dot{\omega}\ (\theta_Q^{\alpha})$.

Observe that $\dot{\omega}$ can be seen as a collection of weightings of $sol^{\mathbb{D}}(Q)$ for $Q \in cqs(L)$. It turns out that evaluating linear expressions and constraints of $\langle L \rangle^{\mathbb{D}}$ with $\dot{\omega}$ returns the same value as the evaluation of linear expressions and constraints of $\rho^{\mathcal{T},\mathbb{D}}(L)$ with $\Pi(\dot{\omega})$:

▶ **Lemma 15.** *For any $\mathcal{T}-$projecting sum $S \in L_{E\Sigma}$ and environment $\gamma : X \to dom(\mathbb{D})$ where $fv(S) \subseteq X$ it holds that $[\![\langle S \rangle^{\mathbb{D},\gamma}]\!]_{\dot{\omega}} = [\![\rho^{\mathcal{T},\mathbb{D},\gamma}(S)]\!]_{\Pi(\dot{\omega})}$.*

▶ **Lemma 16.** *For any constraint $C \in L_{C\Sigma}$ that is $\mathcal{T}-$projecting and environment $\gamma : X \to dom(\mathbb{D})$ where $fv(C) \subseteq X$: $[\![\langle C \rangle^{\mathbb{D},\gamma}]\!]_{\dot{\omega}} = [\![\rho^{\mathcal{T},\mathbb{D},\gamma}(C)]\!]_{\Pi(\dot{\omega})}$.*

Lemma 15 and Lemma 16 rely on Proposition 11. It is easy to see that they imply that if $\dot{\omega}$ is a solution of $\langle L \rangle^{\mathbb{D}}$ (the fact that it respects the local soundness constraints follows from Proposition 11), then $\Pi(\dot{\omega})$ is a solution of $\rho^{\mathcal{T},\mathbb{D}}(L)$ with the same value. Thus, the optimal value of $\rho^{\mathcal{T},\mathbb{D}}(L)$ is greater or equal than the optimal value of $\langle L \rangle^{\mathbb{D}}$.

To prove the equality, it remains to prove that the optimal value of $\langle L \rangle^{\mathbb{D}}$ is greater or equal than the optimal value of $\rho^{\mathcal{T},\mathbb{D}}(L)$. To this end, consider a solution of $\rho^{\mathcal{T},\mathbb{D}}(L)$. It is a weighting $\dot{W}$ of $\Xi_L^{\mathcal{T},\mathbb{D}}$ which respects the local soundness constraints. By Theorem 14, we will be able to reconstruct a weighting $\dot{\omega}$ of Θ_L which respects the constraint of $\langle L \rangle^{\mathbb{D}}$. It is formalized in the following lemma whose proof can be found in the the full version [2].

▶ **Lemma 17.** *For any weighting $\dot{W}$ of $\Xi_L^{\mathcal{T},\mathbb{D}}$ such that $[\![\bigwedge_{Q \in \mathcal{Q}} lsc^{\mathcal{T},\mathbb{D}}(Q)]\!]_{\dot{W}} = 1$, there exists a weighting $\dot{\omega}$ of $\Theta_{\mathcal{Q}}$ such that $\dot{W} = \Pi(\dot{\omega})$.*

▶ **Proposition 18.** *Let $\mathbb{D}$ be a database and $\mathcal{T}$ a collection of decomposition tree. Any $\mathcal{T}-$projecting $LP(CQ)$ program $L = (\textbf{\textit{maximize}}\ S\ \textbf{\textit{subject to}}\ C) \in L_{P\Sigma}$ satisfies that:*

1. *For any solution $\dot{\omega}$ of $\langle L \rangle^{\mathbb{D}}$ there is a solution $\dot{W}$ of $\rho^{\mathcal{T},\mathbb{D}}(L)$ s.t. $[\![\langle S \rangle^{\mathbb{D},\emptyset}]\!]_{\dot{\omega}} = [\![\rho^{\mathcal{T},\mathbb{D}}(S)]\!]_{\dot{W}}$.*
2. *For any solution $\dot{W}$ of $\rho^{\mathcal{T},\mathbb{D}}(L)$ there is a solution $\dot{\omega}$ of $\langle L \rangle^{\mathbb{D}}$ s.t. $[\![\langle S \rangle^{\mathbb{D},\emptyset}]\!]_{\dot{\omega}} = [\![\rho^{\mathcal{T},\mathbb{D}}(S)]\!]_{\dot{W}}$.*

4.5 Treatment of Existential Quantifiers

The previous method of factorized interpretation only works for the $LP(CQ_{qf})_{proj}$ fragment, where conjunctive queries are supposed to be quantifier free. It turns out that one can similarly solve linear programs of $LP(CQ)_{proj}$ programs by applying a simple transformation.

For any $LP(CQ)_{proj}$ program L we can move the existential quantifiers of the conjunctive query into the weight expression as follows, yielding an $LP(CQ_{qf})_{proj}$ program $mvq(L)$: we replace every subexpression $\mathbf{weight}_{\mathbf{x}:Q'}(\exists \mathbf{z}.Q)$ of L, where Q is quantifier free, by $\mathbf{weight}_{\mathbf{xz}:Q'}(Q)$ where $\mathbf{xz}$ is the concatenation of vectors $\mathbf{x}$ and $\mathbf{z}$. We have:

▶ **Theorem 19** (Removing Existential Quantifiers). *For any projecting $LP(CQ)$ program, the $LP(CQ_{qf})_{proj}$ program $mvq(L)$ has the same optimal value as L.*

Observe that we can use this technique for the resource delivery problem L. In $mvq(L)$, there is only one query on variables $(f', o', q, q', b', w', c, c')$. It is easy to see that it has hypertree width 2 since we can construct a tree decomposition having two connected bags $\mathcal{B}(u) = \{f', o', b', q, q'\}$ and $\mathcal{B}(v) = \{f', w', b', c, c'\}$. $\mathcal{B}(u)$ is covered by the first two atoms and $\mathcal{B}(v)$ by the last two. Now, because of the weight expressions, we also need to add a bag for $\{f', w'\}$, $\{w'\}$ and $\{w', b'\}$ which can safely be connected to v, and for $\{f', o'\}$ and $\{b', o'\}$ which can safely be connected to u. It gives a decomposition of L of width 2, showing that the factorized interpretation will have fewer variables than the naive interpretation.

5 Preliminary Experimental Results

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 naive interpretation and the factorized interpretation of the resource delivery problem from the introduction 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 genereated 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. 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 8.

As expected when comparing both linear programs we observed a larger number of constraints (due to the soundness constraints) and a smaller number of variables in the factorized interpretation. While building the naive 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 naive 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 naive 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 naive 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 naive one.

6 Conclusion and Future Work

Our preliminary experiments seem to confirm the efficiency of factorized interpretation, in accordance with our complexity results. More thorough benchmarking is needed to evaluate the practical relevance though. Another direction to explore would be to better integrate our approach into a database engine, in the way it is done by `SolveDB` for example. Finally, 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 $LP(CQ)$ for these optimization problems and study how conjunctive query decompositions could help to improve the efficiency.

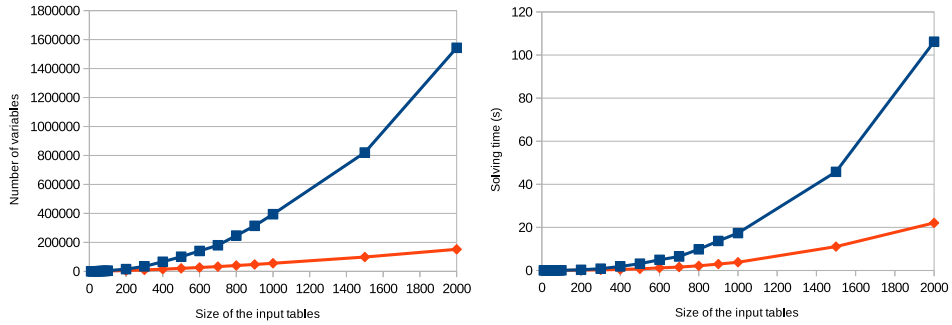

Figure 8 Number of variables and performances of GLPK for naive (blue) and factorized (red) interpretation of the resource delivery problem with respect to table size.

References

1 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.

2 Florent Capelli, Nicolas Crosetti, Joachim Niehren, and Jan Ramon. Linear Programs with Conjunctive Queries. In *25th Internationcal Conference on Database Theory (ICDT 2022)*, Edinburgh, United Kingdom, March 2022. URL: `https://hal.archives-ouvertes.fr/hal-01981553`.

3 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`.

4 Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.

5 Robert Fourer, David M Gay, and Brian W Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, 1990.

6 G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decompositions and Tractable Queries. *Journal of Computer and System Sciences*, 64(3):579–627, May 2002. arXiv: cs/9812022.

7 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.

8 Martin Grohe. The structure of tractable constraint satisfaction problems. In *International Symposium on Mathematical Foundations of Computer Science*, pages 58–72. Springer, 2006.

9 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms (TALG)*, 11(1):4, 2014.

10 Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Comb.*, 4(4):373–396, 1984. `doi:10.1007/BF02579150`.

11 Ton Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994.

12 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, April 2013. `doi:10.14778/2536336.2536341`.

13 Leonid Libkin. *Elements of finite model theory*. Springer Science & Business Media, 2013.

14 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`.

15 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.

16 Dan Olteanu and Jakub Závodnỳ. Factorised representations of query results: size bounds and readability. In *Proceedings of the 15th International Conference on Database Theory*, pages 285–298. ACM, 2012.

17 Dan Olteanu and Jakub Závodnỳ. Size bounds for factorised representations of query results. *ACM Transactions on Database Systems (TODS)*, 40(1):1–44, 2015.

18 Reinhard Pichler and Sebastian Skritek. Tractable counting of the answers to conjunctive queries. *Journal of Computer and System Sciences*, 79:984–1001, September 2013.

19 Laurynas Šikšnys 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.

20 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, November 2013.

21 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.