Linear Programs with Conjunctive Queries

Florent Capelli, Nicolas Crosetti, Joachim Niehren, 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 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 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 hypertree decompositions of small width to group 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 Computer Science, Logic, Databases.

Keywords and phrases Database queries, linear programming, hypergraph decomposition.

Digital Object Identifier 10.4230/LIPIcs...

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 [7] and for constraint programming in MiniZinc [17]. 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 [14], or when using linear optimization to explain the result of a database query to the user as proposed by Meliou and Suciu [16]. Moreover, tools like `SolveDB` [21] 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 any projecting $LP(CQ)$ program over a database to a linear program without conjunctive queries. 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 quantifier are dealt with by showing that one can find an equivalent projecting $LP(CQ)$ program using only quantifier free conjunctive queries.

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. 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:

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

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 questions 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_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 $\mathbf{weight}_{(f', w', b', o'): f' \doteq f \wedge w' \doteq w}(Q)$ 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.

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

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

By answering the query Q on the database $\mathbb{D}$ and introducing a linear program variable θ_Q^α 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 query Q has a hypertree decomposition of width 2 as well as the whole $LP(CQ)$ program.

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 [5] 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 Appendix A.

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. [23] 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 a overall weight less then 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 Appendix B.

1.2 Related Work

Our result builds on well-known techniques using dynamic programming on tree decompositions of the hypergraph of conjunctive queries. This techniques were first introduced by Yannkakis [24] 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 [9, 10, 11] 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 [19] or enumerating it with small delay [1]. More recently, these tractability results have been explained by the mean of factorized databases [18], observing that the answer set of bounded width conjunctive queries could be succinctly represented by circuits enjoying interesting syntactic properties allowing to efficiently solve numerous aggregation problems on it in polynomial time in the size of the representation. While the complexity of several aggregation problems in databases have been studied on such structures [2, 20], our result is, to the best of our knowledge, the first one to exploit the structure of conjunctive queries to solve linear programs more efficiently. While our result could be made to work on factorized representations of queries answer sets, we choose to directly work on tree decompositions because the semantic of the query is clearer in its conjunctive form than its factorized representation. Since one of our contribution is to offer a language to describe linear programs parametrized by the answer set of queries, this aspect is important to write intelligible linear programs.

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 have been moved to the appendix.

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 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 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 $set(\mathbf{x}) = \{x_1, \ldots, x_n\}$ the set of the elements of $\mathbf{x}$. For any variable assignment

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

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

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

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 $r \in \mathcal{R}^{(n)}$.

$\alpha : X \to D$ 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 sums Sum, linear constraints $L\mathrm{C}$, and linear programs $L\mathrm{P}$ 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 **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 Figure 10. 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 real numbers. For any weightings $\omega : \Xi \to \mathbb{R}_+$, the value of a sum $S \in \mathit{Sum}$ 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 [12].

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 $\uparrow u = \{v \in \mathcal{V} \mid (v, u) \in \mathcal{E}^*\}$ the set of nodes that are in the context of or equal to 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 $E \in \mathit{Ex}_{\mathcal{C}}$ is either a (query) variable $x \in \mathcal{X}$ or a constant $a \in \mathcal{C}$.

The set of conjunctive queries $Q \in C_{\mathbb{Q}_\Sigma}$ 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 $fv(Q) \subseteq \mathcal{X}$ 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 $Q \in C_{\mathbb{Q}_\Sigma}$, set $X \supseteq fv(Q)$ and database $\mathbb{D} \in db_\Sigma$ we define the answer set $sol^{\mathbb{D}}_X(Q)$ in Figure 11. It contains all those assignments $\alpha : X \to dom(\mathbb{D})$ for which Q becomes true on $\mathbb{D}$. We also write $sol^{\mathbb{D}}(Q)$ instead of $sol^{\mathbb{D}}_{fv(Q)}(Q)$. Observe that $sol^{\mathbb{D}}(\exists \mathbf{x}.Q) = sol^{\mathbb{D}}(Q)_{|fv(Q) \setminus set(\mathbf{x})}$.

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 the variables of each atom of the query is covered by at least one bag:

▶ **Definition 2** (Hypertree width of quantifier free conjunctive queries)**.** *Let $Q \in C_{\mathbb{Q}_\Sigma}$ 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 $\mathcal{B} = [1/\{x\}, 2/\{x, y\}, 3/\{y, z\}]$ 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** (Folklore)**.** *Given a tree decomposition $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ of a quantifier free conjunctive query $Q \in C_{\mathbb{Q}_\Sigma}$ 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 and can be proven by computing the semi-join of every bag in a subtree in a bottom-up fashion, as it is done in [15, Theorem 6.25]. It gives 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 $O(|\mathbb{D}|^n)$ whereas $(sol^{\mathbb{D}}(Q)_{|\mathcal{B}(u)})_{u \in \mathcal{V}}$, that 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 $n = 1$, 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.

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

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

Figure 4 $LP(CQ)$ programs $L \in LP_{\Sigma}$ where $c \in \mathbb{R}$, $E \in Ex_{\mathcal{C}}$, $\mathbf{x} \in \mathcal{X}^*$ and $Q, Q' \in CQ_{\Sigma}$.

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

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 [13]). *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

3.1 Syntax

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 that we also call linear CQ-programs. 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 $Q, Q' \in CQ_{\Sigma}$. Intuitively, this expressions is interpreted as a linear sum 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 answer set of $Q \wedge Q'$ on the database $\mathbb{D}$ with variables in $set(\mathbf{x})$.

Beside weight expressions, linear sums in Sum_{Σ} may also contain expression $N \in Num_{\mathcal{C}}$ or NS where $S \in Sum_{\Sigma}$ and N is a constant number expression, which is either a real number

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

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

Figure 5 Free variables of linear sums, constraints, and linear CQ-programs.

$c \in \mathbb{R}$ 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.

Linear constrains $C \in L_{\mathrm{C}\Sigma}$ are conjunctions of inequalities $S \leq S'$ between linear sums $S, S' \in Sum_\Sigma$, 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 the $\forall \mathbf{x}{:}Q.\ C'$). The bound variables in $\mathbf{x}$ have scope over Q and C.

$LP(CQ)$ programs or equivalently linear CQ-programs $L \in L_{\mathrm{P}\Sigma}$ are build from linear sums in Sum_Σ and linear constraints in $L_{\mathrm{P}\Sigma}$ as one might expect. Note, however, that free query variables are ruled out at this level, while being permitted in nested linear constraints in $L_{\mathrm{C}\Sigma}$ and linear sums in Sum_Σ.

The sets of free variables of linear sums, constraints, and programs are formally defined in Figure 5. For instance, the following linear 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 a $LP(CQ)$ 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 naïve interpretation from now on.

For doing so, one step is to replace the free query variables of the $LP(CQ)$ programs by elements from the database. For this we assume that we have constants for all elements of the database domain, that is $dom(\mathbb{D}) \subseteq \mathcal{C}$ and define for any conjunctive query Q and variable assignment $\gamma : Y \to D$ where $fv(Q) \subseteq Y$ a conjunctive query $sbs_\gamma(Q)$, by replacing in Q all free occurrences of variables $y \in Y$ by $\gamma(y)$. The formal definition is given in Figure 12.

In order to define the semantics of a linear 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

$$\langle \mathbf{weight}_{\mathbf{x}:Q'}(Q)\rangle^{\mathbb{D},\gamma} = \sum_{\alpha \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q \wedge Q'))} \theta^{\alpha}_{sbs_{\tilde{\gamma}}(Q)} \qquad \langle S_1 + S_2\rangle^{\mathbb{D},\gamma} = \langle S_1\rangle^{\mathbb{D},\gamma} + \langle S_2\rangle^{\mathbb{D},\gamma}$$
$$\langle \forall \mathbf{x}{:}Q.\ C\rangle^{\mathbb{D},\gamma} = \bigwedge_{\gamma' \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q))} \langle C\rangle^{\mathbb{D},\tilde{\gamma}\cup\gamma'} \qquad \langle S_1 \leq S_2\rangle^{\mathbb{D},\gamma} = \langle S_1\rangle^{\mathbb{D},\gamma} \leq \langle S_2\rangle^{\mathbb{D},\gamma}$$
$$\langle \textstyle\sum_{\mathbf{x}:Q} S\rangle^{\mathbb{D},\gamma} = \sum_{\gamma' \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_{\tilde{\gamma}}(Q))} \langle S\rangle^{\mathbb{D},\tilde{\gamma}\cup\gamma'} \qquad \langle C_1 \wedge C_2\rangle^{\mathbb{D},\gamma} = \langle C_1\rangle^{\mathbb{D},\gamma} \wedge \langle C_2\rangle^{\mathbb{D},\gamma}$$
$$\langle NS\rangle^{\mathbb{D},\gamma} = \langle N\rangle^{\mathbb{D},\gamma}\langle S\rangle^{\mathbb{D},\gamma} \qquad \langle true\rangle^{\mathbb{D},\gamma} = true$$
$$\langle \mathbf{num}(a)\rangle^{\mathbb{D},\gamma} = \mathbf{num}^{\mathbb{D}}(a^{\mathbb{D}}) \quad \text{(may be undefined)} \qquad \langle c\rangle^{\mathbb{D},\gamma} = c$$

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

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

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 sum:

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

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

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 Example from Resource Delivery Optimization

Reconsider the $LP(CQ)$ program L from Figure 1 with the following database $\mathbb{D}$:

$$prod^{\mathbb{D}} = \{(F, O_1, 1.5), (F, O_2, 2.2)\} \qquad store^{\mathbb{D}} = \{(W_1, 0.9), (W_2, 2.5)\}$$
$$route^{\mathbb{D}} = \{(F, W_1, 0.5), (F, W_2, 0.4), (W_1, B, 0.6), (W_2, B, 0.8)\} \qquad order^{\mathbb{D}} = \{(B, O_1, 0.8), (B, O_2, 1.4)\}$$

The answer set of query $Q = dlr(f', w', b', o')$ on $\mathbb{D}$ is to be weighted. We denote every answer $\alpha : \{f', w', b', o'\} \to dom(\mathbb{D})$ by $(\alpha(f'), \alpha(w'), \alpha(b'), \alpha(o'))$. Then:

$$sol^{\mathbb{D}}(dlr) = \{(F, W_1, B, O_1), (F, W_2, B, O_1), (F, W_1, B, O_2), (F, W_2, B, O_2)\}$$

The naive interpretation $\langle L\rangle^{\mathbb{D}}$ is the following linear program with variables in $\Theta^{\mathbb{D}}_L$:

$$\begin{aligned}
\mathbf{minimize}\quad & 0.5\ (\theta_Q^{(F,W_1,B,O_1)} + \theta_Q^{(F,W_1,B,O_2)}) + 0.4\ (\theta_Q^{(F,W_2,B,O_1)} + \theta_Q^{(F,W_2,B,O_2)}) \\
& +0.6\ (\theta_Q^{(F,W_1,B,O_1)} + \theta_Q^{(F,W_1,B,O_2)}) + 0.8\ (\theta_Q^{(F,W_2,B,O_1)} + \theta_Q^{(F,W_2,B,O_2)}) \\
\mathbf{subject\ to}\quad & \theta_Q^{(F,W_1,B,O_1)} + \theta_Q^{(F,W_2,B,O_1)} \leq 1.5 \quad \wedge \quad \theta_Q^{(F,W_1,B,O_2)} + \theta_Q^{(F,W_2,B,O_2)} \leq 2.2 \\
\wedge\quad & \theta_Q^{(F,W_1,B,O_1)} + \theta_Q^{(F,W_2,B,O_1)} \geq 0.8 \quad \wedge \quad \theta_Q^{(F,W_1,B,O_2)} + \theta_Q^{(F,W_2,B,O_2)} \geq 1.4 \\
\wedge\quad & \theta_Q^{(F,W_1,B,O_1)} + \theta_Q^{(F,W_1,B,O_2)} \leq 0.9 \quad \wedge \quad \theta_Q^{(F,W_2,B,O_1)} + \theta_Q^{(F,W_2,B,O_2)} \leq 2.5
\end{aligned}$$

The term $(\theta_Q^{(F,W_1,B,O_1)} + \theta_Q^{(F,W_1,B,O_2)})$ in the objective function of this linear program is obtained by computing the value of the expression $\mathbf{weight}_{(f',w',b',o'):f' \doteq s \wedge w' \doteq e}(Q)$ with

the environment $[f'/F, w'/W_1]$. Similarly the term $\theta_Q^{(F,W_1,B,O_1)} + \theta_Q^{(F,W_2,B,O_1)}$ in the first constraint is obtained by computing the value of $\mathbf{weight}_{(f',w',b',o'):f'\doteq f\wedge o'\doteq o}(Q)$ with the environment $[r/O_1, f/F]$. Observe that both weight expressions share the same linear program variable $\theta_Q^{(F,W_1,B,O_1)}$ so they are related semantically.

4 An Efficiently Solvable Fragment

We introduce a class of projecting $LP(CQ)$ programs and define a notion of width of linear CQ-program 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 6.** *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 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 interpreter $\mathbf{weight}_{\mathbf{x}:Q'}(Q)$ 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 thereof.

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 7.** *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*

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

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

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

$\mathcal{B}_Q(u) = set(\mathbf{x}')$. *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 rest of the section is dedicated to proving the following theorem:

▶ **Theorem 8** (Main). *Let* L *be a* $LP(CQ_{qf})_{proj}$ *program,* $\mathcal{T}$ *a decomposition of* L *of width* k *and* $\mathbb{D}$ *a database. There exists an interpretation* $\rho^{\mathcal{T},\mathbb{D}}(L)$ *of* L *having the same value as* $\langle L\rangle^{\mathbb{D}}$ *and* $O(|\mathcal{T}||\mathbb{D}|^k)$ *variables.*

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 8, 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

Based on hypertree decompositions we present an alternative factorized interpretation to smaller linear program having the same optimal value.

In this section, we explain how we can exploit tree decompositions of projecting $LP(CQ)$ programs with quantifier free conjunctive queries to find a smaller interpretation. We fix a program L of $LP(CQ_{qf})_{proj}$. 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 *$\mathcal{T}$-factorized interpretation* $\rho^{\mathcal{T},\mathbb{D}}(L)$ *of* L is formally defined in Figure 7. It mainly mirrors the naïve 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 one can observe the addition *local soundness constraints* $lsc^{\mathcal{T},\mathbb{D}}(Q)$ to the program.

The set of linear program 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 $|\mathcal{T}||\mathbb{D}|^k$ variables in $\rho^{\mathcal{T},\mathbb{D}}(L)$, as stated in Theorem 8.

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.

To prove that $\rho^{\mathcal{T},\mathbb{D}}(L)$ has the same optimal value as $\langle L\rangle^{\mathbb{D}}$, 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 $Q \in cqs(L)$ (where $A = sol^{\mathbb{D}}(Q)$):

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

Observe we added at most $|\mathbb{D}|^k|\mathcal{E}_Q|$ constraints for each $Q \in \mathcal{Q}$. Moreover constructing $\rho^{\mathcal{T},\mathbb{D}}(L)$ from L and $\mathbb{D}$ mainly relies on being able to compute $sol^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$ for every node u of T_Q. This is possible in polynomial time by dynamic programming on T_Q, see Lemma 3.

4.3 Correctness

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 9.** *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 10.** *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 11.** *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 decompositions of Q:

▶ **Proposition 12.** *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 12 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 4.4.

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 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 assignment 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 13 follows by Proposition 10 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 Appendix C.

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 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 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 rely 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 linear 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 sums and constraints of $\langle L\rangle^{\mathbb{D}}$ with $\dot{\omega}$ returns the same value as the evaluation of linear sums and constraints of $\rho^{\mathcal{T},\mathbb{D}}(L)$ with $\Pi(\dot{\omega})$:

▶ **Lemma 14.** *For any $\mathcal{T}$−projecting sum $S \in Sum_\Sigma$ 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 15.** *For any constraint $C \in LC_\Sigma$ 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 14 and Lemma 15 rely on Proposition 10. 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 10), 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 13, 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 appendix.

▶ **Lemma 16.** *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 17.** *Let $\mathbb{D}$ be a database and $\mathcal{T}$ a collection of decomposition tree. Any $\mathcal{T}$−projecting $LP(CQ)$ program $L = (\textbf{\textit{maximize}}\ S\ \textbf{\textit{subject to}}\ C) \in LP_\Sigma$ 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. $[\![\langle S\rangle^{\mathbb{D},\emptyset}]\!]_{\dot{\omega}} = [\![\rho^{\mathcal{T},\mathbb{D}}(S)]\!]_{\dot{W}}$.*

4.4 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 18** (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

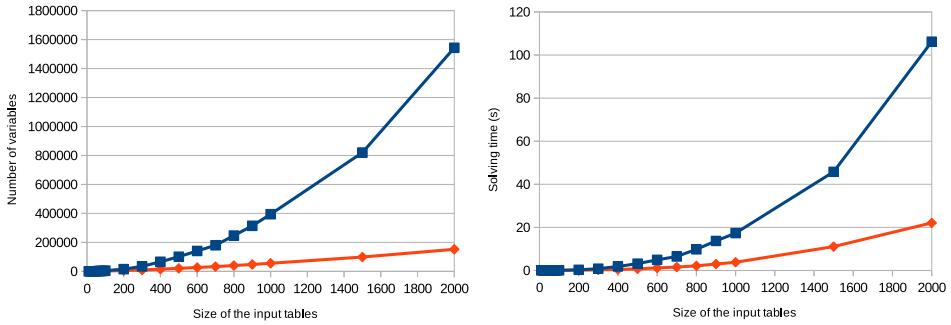

Figure 8 Number of variables and performances of GLPK for naive (blue) and factorized (red) interpretation of the resource delivery problem with respect to table size.

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 factorized interpretation will have less 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. We used Python and the Pulp library to build the linear programs. The tree-decomposition of the *dlr* query is hard-coded. The tests were run on an office laptop by making the number of tuples in the randomly filled *prod*, *order* and *route* tables vary. 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 do 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.

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.

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. Nurzhan Bakibayev, Tomáš Kočiskỳ, Dan Olteanu, and Jakub Závodnỳ. Aggregation and ordering in factorised databases. *Proceedings of the VLDB Endowment*, 6(14):1990–2001, 2013.
3. 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.
4. 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.
5. Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.
6. Mathias Fiedler and Christian Borgelt. Support computation for mining frequent subgraphs in a single graph. In *MLG*. Citeseer, 2007.
7. Robert Fourer, David M Gay, and Brian W Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, 1990.
8. Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.
9. 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.
10. 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.
11. Martin Grohe. The structure of tractable constraint satisfaction problems. In *International Symposium on Mathematical Foundations of Computer Science*, pages 58–72. Springer, 2006.
12. Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Comb.*, 4(4):373–396, 1984. doi:10.1007/BF02579150.
13. Ton Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994.
14. 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. URL: http://dl.acm.org/citation.cfm?doid=2536336.2536341, doi:10.14778/2536336.2536341.
15. Leonid Libkin. *Elements of finite model theory*. Springer Science & Business Media, 2013.
16. 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. URL: http://doi.acm.org/10.1145/2213836.2213875, doi:10.1145/2213836.2213875.
17. 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.
18. Dan Olteanu and Jakub Závodnỳ. Size bounds for factorised representations of query results. *ACM Transactions on Database Systems (TODS)*, 40(1):1–44, 2015.
19. Reinhard Pichler and Sebastian Skritek. Tractable counting of the answers to conjunctive queries. *Journal of Computer and System Sciences*, 79:984–1001, September 2013.
20. Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data*, pages 3–18. ACM, 2016.
21. 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.

22 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.

23 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.

24 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.

A Minimizing Noise in ϵ-Differential Privacy

The strategy of differental privacy is to add noise to the relational data before publication. Roughly speaking, the general objective of ε-differential privacy [5] is to add as few 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 $LP(CQ)$ program with an acyclic conjunctive query.

A.1 Hospital Database about Medical Studies on Patient Tests

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 amout 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:

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

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:

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

Queries

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

Constraints

$$C_{\text{PAT}} = \forall(pat, \varepsilon){:}Priv(pat, \varepsilon).\ \mathbf{weight}_{(pat', test'):pat' \doteq pat}(Q(pat', test')) \leq \mathbf{num}(\varepsilon)$$
$$C_{\text{HOSP}} = \forall(hosp, \varepsilon){:}Priv(hosp, \varepsilon).\ \textstyle\sum_{(pat):H(pat,hosp)} \mathbf{weight}_{(pat', test'):pat' \doteq pat}(InStudy(pat', test')) \leq \mathbf{num}(\varepsilon)$$

Program

$$\textbf{maximize } \textstyle\sum_{(st,test,val):Sens(st,test,val)} \mathbf{num}(val)\ \mathbf{weight}_{(pat', test'):test' \doteq test}(InStudy(pat', test'))$$
$$\textbf{subject to } C_{\text{PAT}} \wedge C_{\text{HOSP}}$$

Figure 9 An $LP(CQ)_{proj}$ program for differential privacy when publishing medical studies aggregating patient tests in hospitals.

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

The $LP(CQ)$ program for this example is given in Figure 9. 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. 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 $LP(CQ)_{proj}$. Furthermore, a hypertree decomposition of width 1 is available. While the naive 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.

B Computing the s-Measure for Graph Pattern Matching

The s-measure has been introduced by Wang et al. [23] 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 maches 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 [3, 4, 6]. A first idea is to count the maximal number of non-overlapping patterns [22]. 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 [8].

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 $h : V_P \to V_G$. 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:

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

We can consider each graph G as a database $\mathbb{D}$ with signature $\Sigma = \{node, edge\}$, domain $dom(\mathbb{D}_G) = V_G$ 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 $V_P = \{1, \ldots, \ell\}$ for some $\ell \in \mathbb{N}$. We can then define a matching of a pattern P by a conjunctive query $match_P(x_1, \ldots, x_\ell)$:

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

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

$$\begin{array}{ll} \textbf{maximize} & \sum_{(x):node(x)} \mathbf{weight}_{(x_1,\ldots,x_n):x_1 \doteq x}(match_P(x_1, \ldots, x_n)) \\ \textbf{subject to} & \forall (x) : node(x) \colon \wedge_{i=1}^{\ell} \mathbf{weight}_{(x_1,\ldots,x_n):x_i \doteq x}(match_P(x_1, \ldots, x_n)) \leq 1. \end{array}$$

Moreover, the hypertree width of the conjunctive query $match_P(x_1, \ldots, x_\ell)$ is at most the (hyper)tree width of the pattern graph P. By our main Theorem 8, 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.

C Weightings

C.1 Projections of weightings

Let $X' \subseteq X \subseteq \mathcal{X}$ and $A \subseteq D^X = \{\alpha \mid \alpha : X \to D\}$ be a set of variable assignments. For any $\alpha' : X' \to D$ we define the set of its extensions into A by $A[\alpha'] = \{\alpha \in A \mid \alpha_{|X'} = \alpha'\}$.

▶ **Lemma 19.** *For any two $\alpha_1, \alpha_2 \in A_{|X'}$, if $\alpha_1 \neq \alpha_2$ then $A[\alpha_1] \cap A[\alpha_2] = \emptyset$.*

Proof. 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 $\gamma_1(x') = \alpha_1(x') \neq \alpha_2(x') = \gamma_2(x')$. ◀

▶ **Lemma 20.** *For $A \subseteq D^X$, $X'' \subseteq X' \subseteq X$, $\alpha'' \in A_{|X''}$: $A[\alpha''] = \biguplus_{\alpha' \in A_{|X'}[\alpha'']} A[\alpha']$.*

Proof. First note that the union on the right is disjoint by Lemma 19.

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 $\alpha \in A[\alpha'']$. ◀

For any weighting ω of A and subset of variables $X' \subseteq X$, we define the projection $\pi_{X'}(\omega) : A_{|X'} \to \mathbb{R}_+$ such that for all $\alpha' \in A_{|X'}$:

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

▶ **Proposition 21.** *For $A \in D^X$, $\omega : A \to \mathbb{R}_+$, $X'' \subseteq X' \subseteq X$: $\pi_{X''}(\omega) = \pi_{X''}(\pi_{X'}(\omega))$.*

Proof sketch. This is a consequence of the disjoint decomposition of Lemma 20. ◀

Proof. Indeed, let $\alpha'' \in A_{|X''}$. We have:

$$\begin{aligned}
\pi_{X''}(\omega)(\alpha'') &= \sum_{\alpha \in A[\alpha'']} \omega(\alpha) && \text{by definition} \\
&= \sum_{\alpha' \in A_{|X'}[\alpha'']} \sum_{\alpha \in A[\alpha']} \omega(\alpha) && \text{by Lemma 20} \\
&= \sum_{\alpha' \in A_{|X'}[\alpha'']} \pi_{X'}(\omega)(\alpha') && \text{by definition of } \pi_{X'}(\omega) \\
&= \pi_{X''}(\pi_{X'}(\omega))(\alpha'') && \text{by definition of } \pi_{X''}(\pi_{X'}(\omega)).
\end{aligned}$$

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

C.2 Weighting collections on decomposition trees

Let $X \subseteq \mathcal{X}$ be a finite set of variables and $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ a decomposition tree of 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 22.** *Let $A \subseteq D^X$ and let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a decomposition tree for X. We call a family $W = (W_v)_{v \in \mathcal{V}}$ a* weighting collection on T for A *if it satisfies the following two 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., $\pi_{\mathcal{B}^{uv}}(W_u) = \pi_{\mathcal{B}^{uv}}(W_v)$.*

▶ **Proposition 23.** *Let $A \subseteq D^X$ and let $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ be a decomposition tree for X. For any weigthing $\omega : A \to \mathbb{R}_+$, 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 Proposition 21 $\pi_{\mathcal{B}^{uv}}(W_u) = \pi_{\mathcal{B}^{uv}}(\omega)$. Similarly $\pi_{\mathcal{B}^{uv}}(W_v) = \pi_{\mathcal{B}^{uv}}(\omega)$. ◀

We next show that the global soundness at any subset of nodes $\{u, v\} \subseteq \mathcal{V}$ follows from the local soundness at all subsets $\{u, v\}$ such that $(u, v) \in \mathcal{E}$.

▶ **Lemma 24.** *If W is sound for T at $\{u, v\}$ for all edges $(u, v) \in \mathcal{E}$ then W is sound for T at all subsets $\{u, v\} \subseteq \mathcal{V}$.*

Proof. We show by induction on $k \geq 0$ for all pairs of nodes $(u, v) \in (\mathcal{E} \cup \mathcal{E}^{-1})^k$ that $\pi_{\mathcal{B}^{uv}}(W_u) = \pi_{\mathcal{B}^{uv}}(W_v)$.

The base case where $k = 0$ is obvious. We now show the induction step from k to $k + 1$. Let $(u, v) \in (\mathcal{E} \cup \mathcal{E}^{-1})^{k+1}$ be arbitrary. Then there exists $w \in \mathcal{V}$ such that $(u, w) \in (\mathcal{E} \cup \mathcal{E}^{-1})^k$ and $(v, w) \in \mathcal{E} \cup \mathcal{E}^{-1}$. By induction hypothesis, we have $\pi_{\mathcal{B}^{uw}}(W_u) = \pi_{\mathcal{B}^{uw}}(W_w)$. We need to show that $\pi_{\mathcal{B}^{uv}}(W_u) = \pi_{\mathcal{B}^{uv}}(W_v)$. We first observe that $\mathcal{B}^{uv} \subseteq \mathcal{B}(w)$ by connectedness of T which implies that $\mathcal{B}^{uv} \subseteq \mathcal{B}^{uw}$ and $\mathcal{B}^{uv} \subseteq \mathcal{B}^{vw}$. Therefore, we can conclude as follows:

$$
\begin{array}{rcll}
\pi_{\mathcal{B}^{uv}}(W_u) & = & \pi_{\mathcal{B}^{uv}}(\pi_{\mathcal{B}^{uw}}(W_u)) & \text{by Proposition 21 and } \mathcal{B}^{uv} \subseteq \mathcal{B}^{uw} \\
& = & \pi_{\mathcal{B}^{uv}}(\pi_{\mathcal{B}^{uw}}(W_w)) & \text{by induction hypothesis} \\
& = & \pi_{\mathcal{B}^{uv}}(W_w) & \text{by Proposition 21 and } \mathcal{B}^{uv} \subseteq \mathcal{B}^{uw} \\
& = & \pi_{\mathcal{B}^{uv}}(\pi_{\mathcal{B}^{vw}}(W_w)) & \text{by Proposition 21 and } \mathcal{B}^{uv} \subseteq \mathcal{B}^{vw} \\
& = & \pi_{\mathcal{B}^{uv}}(\pi_{\mathcal{B}^{vw}}(W_v)) & \text{by local soundness at } \{v, w\} \\
& = & \pi_{\mathcal{B}^{uv}}(W_v) & \text{by Proposition 21 and } \mathcal{B}^{uv} \subseteq \mathcal{B}^{vw}
\end{array}
$$

◀

If T is normalized then the local soundness constraint (22) of W at $(u, v) \in \mathcal{E}$ can be rewritten equivalently into a simpler form as follows:

- if u is an extend node with unique child v then: $\forall \beta \in A_{|\mathcal{B}(v)} : \sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} W_u(\beta') = W_v(\beta)$,
- if u is a project node with unique child v then $\forall \beta \in A_{|\mathcal{B}(u)} : W_u(\beta) = \sum_{\beta' \in A_{|\mathcal{B}(v)}[\beta]} W_v(\beta')$,
- if u is a join node with child v then $\forall \beta \in A_{|\mathcal{B}(u)} : W_u(\beta) = W_v(\beta)$.

C.3 Conjunctive decomposition

We need to restrict ourselves to particular subsets of variable assignments, including answer sets of acyclic conjunctive queries. More generally, we define what it means for a subset of variable assignments to be conjunctively decomposed by a decomposition tree. For any decomposition tree $T = (\mathcal{V}, \mathcal{E}, \mathcal{B})$ and subset $V \subseteq \mathcal{V}$ we define the set of variables:

$$\mathcal{B}(V) = \bigcup_{v \in V} \mathcal{B}(v)$$

In particular, this defines for any $v \in \mathcal{V}$ the union $\mathcal{B}(\uparrow v)$ of bags of vertices in-the-context-or-equal-to v, and the union $\mathcal{B}(\downarrow v)$ of bags of vertices that are descendants-or-equal-to v.

▶ **Definition 25.** *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 holds in general. To see this let $\beta \in A_{|\mathcal{B}(u)}$. If $\alpha \in A[\beta]$ then $\alpha \in A$ and $\beta = \alpha_{|\mathcal{B}(u)}$. Hence, $\alpha = \alpha_{|\mathcal{B}(\uparrow u)} \cup \alpha_{|\mathcal{B}(\downarrow u)}$, so we can define $\alpha_1 = \alpha_{|\mathcal{B}(\uparrow u)}$ and $\alpha_2 = \alpha_{|\mathcal{B}(\downarrow u)}$.

▶ **Proposition 26.** *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.*

Proof. Let u be a node of T. The proof is based on the following observation: given an atom $R(\mathbf{x})$ of Q, either $\mathbf{x} \subseteq \mathcal{B}(\downarrow u)$ or $\mathbf{x} \subseteq \mathcal{B}(\uparrow u)$. Thus Q can be written as $Q_1 \wedge Q_2$ with the variables of Q_1 included in $\mathcal{B}(\downarrow u)$ and the variables of Q_2 includes in $\mathcal{B}(\uparrow u)$. Moreover, recall that $\mathcal{B}(u) = \mathcal{B}(\downarrow u) \cap \mathcal{B}(\uparrow u)$. Thus, given an assignment β of $\mathcal{B}(u)$ and $\alpha|_1 \in sol^{\mathbb{D}}(Q_1)[\beta]$ and $\alpha|_2 \in sol^{\mathbb{D}}(Q_2)[\beta]$, we have that $\alpha = \alpha_1 \cup \alpha_2 \in sol^{\mathbb{D}}(Q)[\beta]$. That is, $sol^{\mathbb{D}}(Q)$ is conjunctively decomposed by T. ◀

▶ **Lemma 27.** *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. ◀

▶ **Lemma 28.** *Let T be a decomposition tree of X, u a join node of T with children $v_1, \ldots, v_k$ 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:*

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

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

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

◀

C.4 Weightings correspondence

We are now ready to prove the main correspondence between weightings of A and weighting collection on T:

▶ **Theorem 29.** *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 assignment 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 $\forall u : W_u = \pi_{\mathcal{B}(u)}(\omega)$.*

▶ **Definition 30.** *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 $u \in \mathcal{V}$, $\omega_u : A_{|\mathcal{B}(\downarrow u)} \to \mathbb{R}_+$ is a weighting defined by induction on well-founded order on the nodes of tree T.

For the base case where u is a leaf of T, we define ω_u such that for all $\alpha \in A_{|\mathcal{B}(\downarrow u)}$:

$$\omega_u(\alpha) = W_u(\alpha)$$

For the induction step we suppose that $\omega_{u'}$ is defined for all children u' of u. With $\beta = \alpha_{|\mathcal{B}(u)}$ we define $\omega_u(\alpha)$ for all $\alpha \in A_{|\mathcal{B}(\downarrow u)}$.

If u is an extend node with a child v then:

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

If u is a project node with a child v then

$$\omega_u(\alpha) = \omega_v(\alpha_{|\mathcal{B}(\downarrow v)}).$$

Observe that $\mathcal{B}(u) = \mathcal{B}(v)$ so $\mathcal{B}(\downarrow u) = \mathcal{B}(\downarrow v)$ thus $\omega_v(\alpha_{|\mathcal{B}(\downarrow v)}) = \omega_v(\alpha)$.

If u is a join node with children $v_1, \dots, v_k$ then

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

▶ **Proposition 31.** *For all $u \in \mathcal{V}$, $W_u = \pi_{\mathcal{B}(u)}(\omega_u)$.*

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

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 $W_v(\beta') = 0$.

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

Recall that by soundness $\sum_{\beta''\in A_{|\mathcal{B}(u)}[\beta']} W_u(\beta'') = W_v(\beta') = 0$. Observe that $\beta \in A_{|\mathcal{B}(u)}[\beta']$ so $W_u(\beta) = 0 = \sum_{\alpha\in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_u(\alpha)$.

Case 1.2 $W_v(\beta') > 0$.

$$\begin{aligned}
&\textstyle\sum_{\alpha\in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_u(\alpha)\\
&= \textstyle\sum_{\alpha\in A_{|\mathcal{B}(\downarrow u)}[\beta]} \frac{W_u(\beta)}{W_v(\beta')}\omega_v(\alpha_{|\mathcal{B}(\downarrow v)}) && \text{by definition}\\
&= \frac{W_u(\beta)}{W_v(\beta')} \textstyle\sum_{\alpha\in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_v(\alpha_{|\mathcal{B}(\downarrow v)})\\
&= \frac{W_u(\beta)}{W_v(\beta')} \textstyle\sum_{\alpha'\in A_{|\mathcal{B}(\downarrow v)}[\beta']} \omega_v(\alpha') && \text{by Lemma 27}\\
&= \frac{W_u(\beta)}{W_v(\beta')} W_v(\beta') && \text{by induction}\\
&= W_u(\beta)
\end{aligned}$$

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

$$\begin{aligned}
&\textstyle\sum_{\alpha\in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_u(\alpha)\\
&= \textstyle\sum_{\alpha\in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_u(\alpha) && \text{by definition}\\
&= \textstyle\sum_{\beta'\in A_{|\mathcal{B}(v)}[\beta]} \sum_{\alpha'\in A_{|\mathcal{B}(\downarrow u)}[\beta']} \omega_v(\alpha) && \text{by Proposition 21 and } \mathcal{B}(v)\subseteq\mathcal{B}(\downarrow u)\\
&= \textstyle\sum_{\beta'\in A_{|\mathcal{B}(v)}[\beta]} W_v(\beta') && \text{by induction and } \mathcal{B}(\downarrow u)=\mathcal{B}(\downarrow v)\\
&= W_u(\beta) && \text{by soundness at } (u,v)
\end{aligned}$$

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

Case 3.2 $W_u(\beta) > 0$.

$$\begin{aligned}
&\textstyle\sum_{\alpha\in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_u(\alpha)\\
&= \textstyle\sum_{\alpha\in A_{|\mathcal{B}(\downarrow u)}[\beta]} \frac{\prod_{i=1}^{k} \omega_{v_i}(\alpha_{|\mathcal{B}(\downarrow v_i)})}{W_u(\beta)^{k-1}} && \text{by definition}\\
&= \textstyle\sum_{\alpha_1\in A_{|\mathcal{B}(\downarrow v_1)}[\beta]} \cdots \sum_{\alpha_k\in A_{|\mathcal{B}(\downarrow v_k)}[\beta]} \frac{\prod_{i=1}^{k} \omega_{v_i}(\alpha_i)}{W_u(\beta)^{k-1}} && \text{by Lemma 28}\\
&= \frac{\prod_{i=1}^{k} \sum_{\alpha_i\in A_{|\mathcal{B}(\downarrow v_i)}[\beta]} \omega_{v_i}(\alpha_i)}{W_u(\beta)^{k-1}}\\
&= \frac{\prod_{i=1}^{k} W_{v_i}(\beta)}{W_u(\beta)^{k-1}} && \text{by induction}\\
&= \frac{W_u(\beta)^{k}}{W_u(\beta)^{k-1}} && \text{by soundness at } (u,v_i)\\
&= W_u(\beta)
\end{aligned}$$

◀

▶ **Lemma 32.** *Let v be the child of an extend node u, $\forall \alpha \in A_{|\mathcal{B}(\downarrow v)}$ with $\beta = \alpha_{|\mathcal{B}(v)}$:*

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

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 $\tau \in A[\alpha]$. ◀

▶ **Lemma 33.** *Given a join node u and its children $v_1,\dots,v_k$, let $\alpha \in A_{|\mathcal{B}(\downarrow v_1)}$ 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 $A_{|\mathcal{B}(\downarrow u)}[\alpha] = \{\tau \in A_{|\mathcal{B}(\downarrow u)}[\beta] \mid \tau_{|\mathcal{B}(\downarrow v_1)} = \alpha\}$ because $\beta = \alpha_{|\mathcal{B}(u)}$.

Thus by Lemma 28, $A_{|\mathcal{B}(\downarrow u)}[\alpha] = \{\tau \in A_{|\mathcal{B}(\downarrow v_1)}[\beta] \bowtie \ldots \bowtie A_{|\mathcal{B}(\downarrow v_k)}[\beta] \mid \tau_{|\mathcal{B}(\downarrow u)} = \alpha\} = \{\alpha\} \bowtie A_{|\mathcal{B}(\downarrow v_2)}[\beta] \bowtie \ldots \bowtie A_{|\mathcal{B}(\downarrow v_k)}[\beta]$ ◀

▶ **Proposition 34.** *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 32, $\sum_{\tau \in A[\alpha]} \omega_r(\tau) = \sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} \sum_{\tau \in A[\alpha \cup \beta']} \omega_r(\tau)$. By induction this is equal to $\sum_{\beta' \in A_{|\mathcal{B}(u)}[\beta]} \omega_u(\alpha \bowtie \beta')$.

Case 1.1 $W_v(\beta) = 0$.

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

Observe that by Proposition 31, $\sum_{\alpha' \in A_{|\mathcal{B}(\downarrow u)}[\beta]} \omega_v(\alpha') = W_v(\beta) = 0$. However ω_v is non-negative so $\omega_v(\alpha) = 0 = \sum_{\tau \in A[\alpha]} \omega_r(\tau)$.

Case 1.2 $W_v(\beta) > 0$.

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

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

By Proposition 21, $\sum_{\tau \in A[\alpha]} \omega_r(\tau) = \sum_{\alpha' \in A_{|\mathcal{B}(\downarrow u)}[\alpha]} \sum_{\tau \in A[\alpha']} \omega_r(\tau)$.

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

Case 3.1 $W_u(\beta) = 0$.

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, $W_v(\beta) = W_u(\beta) = 0$ so similarly to Case 1.2, $\omega_v(\alpha) = 0 = \sum_{\tau \in A[\alpha]} \omega_r(\tau)$.

Case 3.2 $W_v(\beta) > 0$.

By definition of ω_u, $\sum_{\alpha' \in A_{|\mathcal{B}(\downarrow u)}[\alpha]} \omega_u(\alpha') = \sum_{\alpha' \in A_{|\mathcal{B}(\downarrow u)}[\alpha]} \frac{\prod_{i=1}^{k} \omega_{v_i}(\alpha'_{|\mathcal{B}(\downarrow v_i)})}{W_u(\beta)^{k-1}}$. Moreover by Lemma 33 we can split α' into $\alpha \times \alpha_2 \times \cdots \times \alpha_k$ and the sum into $\sum_{\alpha_2 \in A_{|\mathcal{B}(\downarrow v_2)}[\beta]} \cdots \sum_{\alpha_k \in A_{|\mathcal{B}(\downarrow v_k)}[\beta]} \frac{\prod_{i=1}^{k} \omega_{v_i}(\alpha'_{|\mathcal{B}(\downarrow v_i)})}{W_u(\beta)^{k-1}}$. 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 $\omega_v(\alpha) \cdot \frac{\prod_{i=2}^{k} \sum_{\alpha_i \in A_{|\mathcal{B}(\downarrow v_i)}[\beta]} \omega_{v_i}(\alpha_i)}{W_u(\beta)^{k-1}}$ which is equal to $\omega_v(\alpha) \cdot \frac{\prod_{i=2}^{k} W_{v_i}(\beta)}{W_u(\beta)^{k-1}}$ by Proposition 31.

Finally observe that by soundness, $\prod_{i=2}^{k} W_{v_i}(\beta) = W_u(\beta)^{k-1}$.

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

◀

of Theorem 29. The first item follows directly from Proposition 23.

For the second item we know by Proposition 31 that $W_u = \pi_{\mathcal{B}(u)}(\omega_u)$ which is equal to $\pi_{\mathcal{B}(u)}(\pi_{\mathcal{B}(\downarrow u)}(\omega_r))$ by Proposition 34. Thus $W_u = \pi_{\mathcal{B}(u)}(\omega_r)$ by Proposition 21. ◀

$$
\begin{array}{ll}
[\![c]\!]_\omega & = c \\
[\![\xi]\!]_\omega & = \omega(\xi) \\
[\![cS]\!]_\omega & = c \cdot [\![S]\!]_\omega \\
[\![S + S']\!]_\omega & = [\![S]\!]_\omega + [\![S']\!]_\omega \\
 & \\
[\![true]\!]_\omega & = 1 \\
[\![S \leq S']\!]_\omega & = \begin{cases} 1 & \text{if } [\![S]\!]_\omega \leq [\![S']\!]_\omega \\ 0 & \text{otherwise.} \end{cases} \\
[\![C \wedge C']\!]_\omega & = [\![C]\!]_\omega \wedge [\![C']\!]_\omega \\
 & \\
[\![\textbf{maximize } S \textbf{ subject to } C]\!] & = max(\{[\![S]\!]_\omega \mid \omega : \Xi \to \mathbb{R}_+, [\![C]\!]_\omega \doteq 1\})
\end{array}
$$

Figure 10 Evaluation of linear sums, constraints and programs.

$$
\begin{array}{rcll}
eval^{\mathbb{D},\alpha}(x) & = & \alpha(x) & \\
eval^{\mathbb{D},\alpha}(a) & = & a^{\mathbb{D}} & \\
sol^{\mathbb{D}}_X(E_1 \doteq E_2) & = & \{\alpha : X \to D \mid eval^{\mathbb{D},\alpha}(E_1) = eval^{\mathbb{D},\alpha}(E_2)\} & \\
sol^{\mathbb{D}}_X(r(E_1, \ldots, E_n)) & = & \{\alpha : X \to D \mid (eval^{\mathbb{D},\alpha}(E_1), \ldots, eval^{\mathbb{D},\alpha}(E_n)) \in r^{\mathbb{D}}\} & \\
sol^{\mathbb{D}}_X(Q_1 \wedge Q_2) & = & sol^{\mathbb{D}}_X(Q_1) \cap sol^{\mathbb{D}}_X(Q_2) & \\
sol^{\mathbb{D}}_X(\exists x.Q) & = & \{\alpha_{|X} \mid \alpha \in sol^{\mathbb{D}}_{X \cup \{x\}}(Q)\} & \text{if } x \notin X \\
sol^{\mathbb{D}}_X(true) & = & X^{dom(\mathbb{D})} &
\end{array}
$$

Figure 11 Answer sets of conjunctive queries.

A Proofs for Section 2 (Preliminaries)

B Proofs for Section 3 (Linear Programs with Conjunctive Queries)

B.1 α-Renaming may change the semantics of linear Cq-programs.

To illustrate this let $\mathbf{x} = (x_1, x_2)$ and L be the following linear CQ-program:

$$\textbf{maximize } \textbf{weight}_{\mathbf{x}:x_2 \doteq a}(r(\mathbf{x})) \textbf{ subject to } \textbf{weight}_{\mathbf{x}}(r(\mathbf{x})) \leq 1.$$

Let $\mathbb{D}$ be a database with signature $\Sigma = \{r^{(2)}\}$ and interpretation $r^{\mathbb{D}} = \{(0,0),(0,1)\}$. If Q is the query $r(\mathbf{x})$ then the semantics of this linear CQ-program $\langle L \rangle^{\mathbb{D}}$ is the linear program:

$$\textbf{maximize } \theta_Q^{(0,1)} \textbf{ subject to } \theta_Q^{(0,0)} + \theta_Q^{(0,1)} \leq 1$$

The optimal value $\langle L \rangle^{\mathbb{D}}$ is $[\![\langle L \rangle^{\mathbb{D}}]\!] = 1$ since $\theta_Q^{(0,0)} + \theta_Q^{(0,1)} \leq 1$ and $\theta_Q^{(0,0)} \geq 0$. Now let us α-rename the second occurrence of $\mathbf{x}$ in L apart to $\mathbf{x}' = (x_1', x_2')$ yielding the following linear CQ-program L':

$$\textbf{maximize } \textbf{weight}_{\mathbf{x}:x_2 \doteq a}(r(\mathbf{x})) \textbf{ subject to } \textbf{weight}_{\mathbf{x}'}(r(\mathbf{x}')) \leq 1.$$

The semantics $\langle L \rangle^{\mathbb{D}}$ is the following linear program where Q' is $r(\mathbf{x}')$ and Q is $r(\mathbf{x})$:

$$\textbf{maximize } \theta_Q^{(0,1)} \textbf{ subject to } \theta_{Q'}^{(0,0)} + \theta_{Q'}^{(0,1)} \leq 1$$

The optimal value of $\langle L' \rangle^{\mathbb{D}}$ is ∞ since $\theta_Q^{(0,1)}$ is no more constrained: the renaming made the variables of Q and Q' different, so that the variable assignments answering these queries

$$
\begin{array}{rcl}
sbs_\gamma(Q \wedge Q') & = & sbs_\gamma(Q) \wedge sbs_\gamma(Q') \\
sbs_\gamma(\exists x.Q) & = & \exists x.sbs_{\gamma_{|dom(\gamma)\setminus\{x\}}}(Q) \\
sbs_\gamma(r(t_1,\ldots,t_n)) & = & r(sbs_\gamma(t_1),\ldots,sbs_\gamma(t_n)) \\
sbs_\gamma(x) & = & \begin{cases} \gamma(x) & \text{if } x \in dom(\gamma) \\ x & \text{otherwise} \end{cases} \\
sbs_\gamma(a) & = & a
\end{array}
$$

Figure 12 Lifting substitutions $\gamma : fv(Q) \to \mathcal{C}$ to queries Q.

become different too. Therefore, different weights may be assigned to them. In other word the tables of answers of Q and Q' are different, since their columns are named by differently, by $\mathbf{x}$ and respectively $\mathbf{x}'$.

C Proofs for Section 4 (An Efficiently Solvable Fragment)

▶ **Lemma 14.** *For any $\mathcal{T}-$projecting sum $S \in Sum_\Sigma$ and environment $\gamma : X \to dom(\mathbb{D})$ where $fv(S) \subseteq X$ it holds that $[\![\langle S\rangle^{\mathbb{D},\gamma}]\!]_{\dot\omega} = [\![\rho^{\mathcal{T},\mathbb{D},\gamma}(S)]\!]_{\Pi(\dot\omega)}$.*

Proof. By induction on the structure of S.

Case $S = \mathbf{weight}_{\mathbf{x}:\mathbf{x}'\doteq\mathbf{y}}(Q)$ where $set(\mathbf{x}') \subseteq set(\mathbf{x}) = fv(Q)$, $set(\mathbf{y}) \cap set(\mathbf{x}) = \emptyset$ and $set(\mathbf{x}') = \mathcal{B}_Q(u)$ for some node u of T_Q.

Let $\beta = [\mathbf{x}'/\gamma(\mathbf{y})]$.

$$
\begin{array}{rcll}
[\![\langle S\rangle^{\mathbb{D},\gamma}]\!]_{\dot\omega} & = & [\![\sum_{\alpha\in sol^{\mathbb{D}}_{set(\mathbf{x})}(Q\wedge sbs_\gamma(Q'))} \theta^\alpha_Q]\!]_{\dot\omega} & \\
& = & [\![\sum_{\alpha\in sol^{\mathbb{D}}_{set(\mathbf{x})}(Q\wedge\bigwedge_{j=1}^m x'_i=\gamma(y_i))} \theta^\alpha_Q]\!]_{\dot\omega} & \\
& = & [\![\sum_{\substack{\alpha\in sol^{\mathbb{D}}_{set(\mathbf{x})}(Q) \\ \alpha_{|\mathcal{B}(u)}=\beta}} \theta^\alpha_Q]\!]_{\dot\omega} & \text{since } set(\mathbf{x}') = \mathcal{B}_Q(u) \\
& = & \sum_{\substack{\alpha\in sol^{\mathbb{D}}_{set(\mathbf{x})}(Q) \\ \alpha_{|\mathcal{B}(u)}=\beta}} \dot\omega\,(\theta^\alpha_Q) & \\
& = & \sum_{\alpha\in sol^{\mathbb{D}}(Q)[\beta]} \dot\omega\,(\theta^\alpha_Q) &
\end{array}
$$

We distinguish two cases depending on whether $\beta \in sol^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$:

- If $\beta \notin sol^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$ then by definition of $\rho^{\mathcal{T},\mathbb{D},\gamma}(S)$:

$$[\![\langle S\rangle^{\mathbb{D},\alpha}]\!]_{\dot\omega} = \sum_{\alpha\in sol^{\mathbb{D}}(Q)[\beta]} \dot\omega\,(\theta^\alpha_Q) = \sum_{\alpha\in\emptyset} \dot\omega\,(\theta^\alpha_Q) = 0 = \rho^{\mathcal{T},\mathbb{D},\gamma}(S)$$

.

- If $\beta \in sol^{\mathbb{D}}(Q)_{|\mathcal{B}(u)}$ then $\xi_{Q,u,\beta} \in \Xi^{\mathcal{T},\mathbb{D}}_L$ and thus by definition of $\rho^{\mathcal{T},\mathbb{D},\gamma}(S)$:

$$
\begin{array}{rcll}
[\![\langle S\rangle^{\mathbb{D},\gamma}]\!]_{\dot\omega} & = & \sum_{\gamma\in sol^{\mathbb{D}}(Q)[\beta]} \dot\omega\,(\theta^\gamma_Q) & \\
& = & \Pi(\dot\omega)(\xi_{Q,u,\beta}) & \text{by definition of } \Pi \\
& = & [\![\xi_{Q,u,\beta}]\!]_{\Pi(\dot\omega)} & \\
& = & [\![\rho^{\mathcal{T},\mathbb{D},\gamma}(S)]\!]_{\Pi(\dot\omega)} &
\end{array}
$$

Case $S = N$.

Straightforward.

Case $S = \sum_{\mathbf{x}:Q} S'$.

$$\begin{aligned}
[\![\langle S\rangle^{\mathbb{D},\gamma}]\!]_{\dot\omega} &= [\![\sum_{\gamma' \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_\gamma(Q))} \langle S'\rangle^{\mathbb{D},\gamma\cup\gamma'}]\!]_{\dot\omega} \\
&= \sum_{\gamma' \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_\gamma(Q))} [\![\langle S'\rangle^{\mathbb{D},\gamma\cup\gamma'}]\!]_{\dot\omega} \\
&= \sum_{\gamma' \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_\gamma(Q))} [\![\rho^{\mathcal{T},\mathbb{D},\gamma\cup\gamma'}(S')]\!]_{\Pi(\dot\omega)} \quad \text{by induction hyp.} \\
&= [\![\sum_{\gamma' \in sol^{\mathbb{D}}_{set(\mathbf{x})}(sbs_\gamma(Q))} \rho^{\mathcal{T},\mathbb{D},\gamma\cup\gamma'}(S')]\!]_{\Pi(\dot\omega)} \\
&= [\![\rho^{\mathcal{T},\mathbb{D},\gamma}(S)]\!]_{\Pi(\dot\omega)}
\end{aligned}$$

Case $S = NS$. Straightforward.
Case $S = S' + S''$.
Straightforward.

◀

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

Proof. By induction on the structure of C.

Base case 1 $C = true$ Obvious.
Base case 2 $C = S \leq S'$ Straightforward using Lemma 14.
Induction step 1 $C = C' \wedge C''$
Straightforward.
Induction step 2 $C = \forall \mathbf{x}{:}r(\mathbf{x}).C$

$$\begin{aligned}
[\![\langle C\rangle^{\mathbb{D},\gamma}]\!]_{\dot\omega} &= [\![\langle \forall \mathbf{x}{:}r(\mathbf{x}).\ C'\rangle^{\mathbb{D},\gamma}]\!]_{\dot\omega} \\
&= [\![\bigwedge_{\gamma' \in sol^{\mathbb{D}}_{set(\mathbf{x})}(r(\mathbf{x}))} \langle C'\rangle^{\mathbb{D},\gamma\cup\gamma'}]\!]_{\dot\omega} \\
&= \bigwedge_{\gamma' \in sol^{\mathbb{D}}_{set(\mathbf{x})}(r(\mathbf{x}))} [\![\langle C'\rangle^{\mathbb{D},\gamma\cup\gamma'}]\!]_{\dot\omega} \\
&= \sum_{\gamma' \in sol^{\mathbb{D}}_{set(\mathbf{x})}(r(\mathbf{x}))} [\![\rho^{\mathcal{T},\mathbb{D},\gamma\cup\gamma'}(C')]\!]_{\Pi(\dot\omega)} \quad \text{by induction} \\
&= [\![\bigwedge_{\gamma' \in sol^{\mathbb{D}}(r(\mathbf{x}))} \rho^{\mathcal{T},\mathbb{D},\gamma\cup\gamma'}(C')]\!]_{\Pi(\dot\omega)} \\
&= [\![\rho^{\mathcal{T},\mathbb{D},\gamma}(\forall \mathbf{x}{:}r(\mathbf{x}).C')]\!]_{\Pi(\dot\omega)} \\
&= [\![\rho^{\mathcal{T},\mathbb{D},\gamma}(C)]\!]_{\Pi(\dot\omega)}
\end{aligned}$$

◀

▶ **Lemma 16.** *For any weighting $\dot W$ of $\Xi^{\mathcal{T},\mathbb{D}}_L$ 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 $\dot W = \Pi(\dot\omega)$.*

Proof. For each $Q \in \mathcal{Q}$, let $W^Q = (W^Q_u)_{u \in T_Q}$ where $W^Q_u(\beta) = \dot{W}(\xi_{Q,u,\beta})$ for each $\xi_{Q,u,\beta} \in \Xi^{\mathcal{T},\mathbb{D}}_L$.

Observe that it follows from the hypothesis that $[\![lsc^{\mathcal{T},\mathbb{D}}(Q)]\!]_{\dot{W}} = 1$ so given a pair of nodes (u,v) in T_Q and a $\gamma \in sol^{\mathbb{D}}(Q)_{|\mathcal{B}^{uv}}$ then $[\![\sum_{\beta \in A_{|\mathcal{B}(u)}[\gamma]} \xi_{Q,u,\beta}]\!]_{\dot{W}} = [\![\sum_{\beta' \in A_{|\mathcal{B}(v)}[\gamma]} \xi_{Q,v,\beta'}]\!]_{\dot{W}}$. By definition of W^Q we then have $\sum_{\beta \in A_{|\mathcal{B}(u)}[\gamma]} W^Q_u(\beta) = \sum_{\beta' \in A_{|\mathcal{B}(v)}[\gamma]} W^Q_u(\beta')$. By definition of the projection of weightings it follows that $\pi_{\mathcal{B}^{uv}}(W^Q_u) = \pi_{\mathcal{B}^{uv}}(W^Q_v)$ so W^Q is a weighting collection on T_Q. Thus by Theorem 13 for each $Q \in \mathcal{Q}$, there is a weighting $\omega_Q : sol^{\mathbb{D}}(Q) \to \mathbb{R}_+$ such that $W_Q = (\pi_{\mathcal{B}(u)}(\omega_Q))_{u \in \mathcal{V}}$. Finally we define a weighting $\dot{\omega}\colon \Theta_{\mathcal{Q}} \to \mathbb{R}_+$ such that $\dot{\omega}(\theta^\alpha_Q) = \omega_Q(\alpha)$ for each $\theta^\alpha_Q \in \Theta_{\mathcal{Q}}$.

We fix a $\xi_{Q,u,\beta} \in \Xi^{\mathcal{T},\mathbb{D}}_L$. By definition $W(\xi_{Q,u,\beta}) = W^Q_u(\beta) = (\pi_{\mathcal{B}(u)}(\omega_Q))(\beta) = \sum_{\alpha \in sol^{\mathbb{D}}(Q)[\beta]} \omega_Q(\alpha)$. Thus $W(\xi_{Q,u,\beta}) = \Pi(\dot{\omega})(\xi_{Q,u,\beta})$ by definition of Π. ◀

▶ **Proposition 17.** *Let $\mathbb{D}$ be a database and $\mathcal{T}$ a collection of decomposition tree. Any $\mathcal{T}-$projecting $LP(CQ)$ program $L = (\textbf{\textit{maximize}}\ S\ \textbf{\textit{subject to}}\ C) \in LP_\Sigma$ 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. $[\![\langle S \rangle^{\mathbb{D},\emptyset}]\!]_{\dot{\omega}} = [\![\rho^{\mathcal{T},\mathbb{D}}(S)]\!]_{\dot{W}}$.*

Proof. Let $\mathcal{Q} = cqs(L)$.

Case 1 Consider a solution $\dot{\omega}$ of $\langle L \rangle^{\mathbb{D}}$.

Let $\dot{W}$ be a weighting of $\Theta_{\mathcal{Q}}$ such that $\dot{W} = \Pi(\dot{\omega})$. By hypothesis $[\![\langle C \rangle^{\mathbb{D},\emptyset}]\!]_{\dot{\omega}} = 1$ so $[\![\rho^{\mathcal{T},\mathbb{D}}(C)]\!]_{\dot{W}} = 1$ by Lemma 15. Thus $\dot{W}$ is a solution of $\rho^{\mathcal{T},\mathbb{D}}(L)$. Moreover $[\![\langle S \rangle^{\mathbb{D},\emptyset}]\!]_{\dot{\omega}} = [\![\rho^{\mathcal{T},\mathbb{D}}(S)]\!]_{\dot{W}}$ by Lemma 14.

Case 2 Fixed solution $\dot{W}$ of $\rho^{\mathcal{T},\mathbb{D}}(L)$

By Lemma 16 there is a weighting $\dot{\omega}$ of $\Theta_{\mathcal{Q}}$ such that $\dot{W} = \Pi(\dot{\omega})$ By hypothesis $[\![\rho^{\mathcal{T},\mathbb{D}}(C)]\!]_{\dot{W}} = 1$ so $[\![\langle C \rangle^{\mathbb{D},\emptyset}]\!]_{\dot{\omega}} = 1$ by Lemma 15. Thus $\dot{W}$ is a solution of $\rho^{\mathcal{T},\mathbb{D}}(L)$. Moreover $[\![\langle S \rangle^{\mathbb{D},\emptyset}]\!]_{\dot{\omega}} = [\![\rho^{\mathcal{T},\mathbb{D}}(S)]\!]_{\dot{W}}$ by Lemma 14.

◀

▶ **Theorem 18** (Removing Existential Quantifiers). *For any projecting $LP(CQ)$ program, the $LP(CQ_{qf})_{proj}$ program $mvq(L)$ has the same optimal value as L.*

Proof. It is clear that every Q appearing in a subexpression $\textbf{weight}_{\mathbf{xz}:Q'}(Q)$ of $mvq(L)$ is quantifier free by definition. Now, since L is in $LP(CQ)_{proj}$, Q' is of the form $\mathbf{x}' = \mathbf{y}$ where $\mathbf{x}'$ only contains free variables of $\exists \mathbf{z}.Q$. Since $fv(\exists \mathbf{z}.Q) \subseteq fv(Q)$, we have that $\mathbf{x}'$ only contains free variables of Q. Moreover, the other condition of $LP(CQ)_{proj}$ concerning the sum and universal quantification are still respected in $mvq(L)$, thus L is in $LP(CQ_{qf})_{proj}$.

Now, let $\dot{\omega}\colon \Theta^{\mathbb{D}}_L \to \mathbb{R}_+$ be a solution of L. We define $\dot{\omega}'\colon \Theta^{\mathbb{D}}_{mvq(L)} \to \mathbb{R}_+$ as follows: $\dot{\omega}'(\theta^\alpha_Q) = \frac{1}{N}\dot{\omega}(\theta^{\alpha|_U}_{\exists \mathbf{z}.Q})$ where $U = fv(\exists \mathbf{z}.Q)$ and $N = \#\{\beta : \mathbf{z} \to dom \mid \alpha \cup \beta \in sol^{\mathbb{D}}(Q)\}$. It is readily verified that the value $[\![\textbf{weight}_{\mathbf{x}:Q'}(\exists \mathbf{z}.Q)]\!]_{\dot{\omega}}$ is the same as $[\![\textbf{weight}_{\mathbf{x},\mathbf{z}:Q'}(Q)]\!]_{\dot{\omega}'}$ and thus that $\dot{\omega}'$ is a solution of $mvq(L)$ and $[\![L]\!]_{\dot{\omega}} = [\![mvq(L)]\!]_{\dot{\omega}'}$.

For the other way around, given $\dot{\omega}'\colon \Theta^{\mathbb{D}}_{mvq(L)} \to \mathbb{R}_+$ a solution of $mvq(L)$, we construct $\dot{\omega}\colon \Theta^{\mathbb{D}}_L \to \mathbb{R}_+$ as $\dot{\omega}(\theta^\alpha_{\exists \mathbf{z}.Q}) = \sum_{\beta | \alpha \cup \beta \in sol^{\mathbb{D}}(Q)} \dot{\omega}'(\theta^{\alpha \cup \beta}_Q)$. Again, it is readily verified that the value $[\![\textbf{weight}_{\mathbf{x}:Q'}(\exists \mathbf{z}.Q)]\!]_{\dot{\omega}}$ is the same as $[\![\textbf{weight}_{\mathbf{x},\mathbf{z}:Q'}(Q)]\!]_{\dot{\omega}'}$ and thus that $\dot{\omega}$ is a solution of L and $[\![L]\!]_{\dot{\omega}} = [\![mvq(L)]\!]_{\dot{\omega}'}$. ◀