Cost-Driven Ontology-Based Data Access (Extended Version)

Davide Lanti, Guohui Xiao, Diego Calvanese
Faculty of Computer Science
Free University of Bozen-Bolzano, Bolzano, Italy
{dlanti,xiao,calvanese}@inf.unibz.it

Abstract

In *ontology-based data access* (OBDA), users are provided with a conceptual view of a (relational) data source that abstracts away details about data storage. This conceptual view is realized through an *ontology* that is connected to the data source through declarative *mappings*, and query answering is carried out by translating the user queries over the conceptual view into SQL queries over the data source. Standard translation techniques in OBDA try to transform the user query into a union of conjunctive queries (UCQ), following the heuristic argument that UCQs can be efficiently evaluated by modern relational database engines. In this work, we show that translating to UCQs is not always the best choice, and that, under certain conditions on the interplay between the ontology, the mappings, and the statistics of the data, alternative translations can be evaluated much more efficiently. To find the best translation, we devise a cost model together with a novel cardinality estimation that takes into account all such OBDA components. Our experiments confirm that *(i)* alternatives to the UCQ translation might produce queries that are orders of magnitude more efficient, and *(ii)* the cost model we propose is faithful to the actual query evaluation cost, and hence is well suited to select the best translation.

1 Introduction

An important research problem in Big Data is how to provide end-users with effective access to the data, while relieving them from being aware of details about how the data is organized and stored. The paradigm of Ontology-based Data Access (OBDA) [19] addresses this problem by presenting to the end-users a convenient view of the data stored in a relational database. This view is in the form of a *virtual RDF graph* [16] that can be queried through *SPARQL* [13]. Such virtual RDF graph is realized by means of the *TBox of an OWL 2 QL ontology* [18] that is connected to the data source through declarative *mappings*. Such mappings associate to each predicate in the TBox (i.e., a concept, role, or attribute) a SQL query over the data source, which intuitively specifies how to populate the predicate from the data extracted by the query.

Query answering in this setting is not carried out by actually materialising the data according to the mappings, but rather by first *rewriting* the user query with respect to the TBox, and then *translating* the rewritten query into a SQL query over the data. In state-of-the-art OBDA systems [6], such SQL translation is the result of *structural optimizations*, which aim at obtaining a *union of conjunctive queries* (UCQ). Such an approach is claimed to be effective because *(i)* joins are over database values, rather than over URIs constructed by applying mapping definitions; *(ii)* joins in UCQs are performed by directly accessing (usually, indexed) database tables, rather than materialized and non-indexed intermediate views. However, the requirement of generating UCQs comes at the cost of an exponential blow-up in the size of the user query.

A more subtle, *sometimes* critical issue, is that the UCQ structure accentuates the problem of redundant data, which is particularly severe in OBDA where the focus is on retrieving *all* the answers implied by the data and the TBox: each CQ in the UCQ can be seen as a different attempt of enriching the set of retrieved answers, without any guarantee on whether the attempt will be successful in retrieving new results. In fact, it was already observed in [2] that generating UCQs is sometimes counter-beneficial (although that work was focusing on a substantially different topic).

As for the rewriting step, Bursztyn et al. [3, 4] have investigated a space of alternatives to UCQ perfect rewritings, by considering *joins of UCQs* (JUCQs), and devised a cost-based algorithm to select the best alternative. However, the scope of their work is limited to the simplified setting in which there are no mappings and the extension of the predicates in the ontology is directly stored in the database. Moreover, they use their algorithm in combination with traditional cost models from the database literature of query evaluation costs, which, according to their experiments, provide estimations close to the native ones of the PostgreSQL database engine.

In this work we study the problem of alternative translations in the general setting of OBDA, where the presence of mappings needs to be taken into account. To do so, we first study the problem of translating JUCQ perfect rewritings,

such as those from [3], into SQL queries that preserve the JUCQ structure while maintaining property *(i)* above, i.e., the ability of performing joins over database values, rather than over constructed URIs. We also devise a cost model based on a *novel cardinality estimation*, for estimating the cost of evaluating a translation of a UCQ or JUCQ over the database. The novelty in our cardinality estimation is that it exploits the interplay between the components of an OBDA instance, namely ontology, mappings, and statistics of the data, so as to better estimate the number of non-duplicate answers.

We carry out extensive and in-depth experiments based on a synthetic scenario built on top of the *Winsconsin Benchmark* [9], a widely adopted benchmark for databases, so as to understand the trade-off between a translation of UCQs and JUCQs. In these experiments we observe that: *(i)* factors such as the number of mapping assertions, also affected by the number of axioms in the ontology, and the number of redundant answers are the main factors for deciding which translation to choose; *(ii)* the cost model we propose is faithful to the actual query evaluation cost, and hence is well suited to select the best alternative translation of the user query; *(iii)* the cost model implemented by PostgreSQL performs surprisingly poorly in the task of estimating the best translation, and is significantly outperformed by our cost model. The main reason for this is that PostgreSQL fails at recognizing when different translations are actually equivalent, and may provide for them cardinality estimations that differ by several orders of magnitude.

In addition, we carry out an evaluation on a real-world scenario based on the NPD benchmark for OBDA [17]. Also in these experiments we confirm that alternative translations to the UCQ one may be more efficient, and that the same factors already identified in the Winsconsin experiments determine which choice is best.

The rest of the paper is structured as follows. Section 2 introduces the relevant technical notions underlying OBDA. Section 3 place our paper with respect to the field literature. Section 4 provides our characterization for SQL translations of JUCQs. Section 5 presents our novel model for cardinality estimation, and Section 6 the associated cost model. Section 7 provides the evaluation of the cost model on the Wisconsin and NPD Benchmarks. Section 8 concludes the paper.

2 Preliminaries

In this section we introduce basic notions and notation necessary for the technical development of this work.

From now on, we will denote tuples in **bold** font, e.g., $\mathbf{x}$ is a tuple, and when convenient we treat tuples as sets. Given a predicate symbol P, a tuple $\mathbf{f}$ of function symbols, and a tuple $\mathbf{x}$ of variables, we denote by $P(\mathbf{f}(\mathbf{x}))$ an atom where each argument is of the form $g(\mathbf{y})$, where $g \in \mathbf{f}$ and $\mathbf{y} \subseteq \mathbf{x}$.

We rely on the OBDA framework of [19], which we formalize here through the notion of *OBDA specification*, which is a triple $\mathcal{S} = (\mathcal{T}, \mathcal{M}, \Sigma)$ where $\mathcal{T}$ is an *ontology TBox*, $\mathcal{M}$ is a set of *mappings*, and Σ is the schema of a relational database. In the remainder of this section, we formally introduce such elements of an OBDA specification.

We assume that ontologies are formulated in *DL-Lite*$_\mathcal{R}$ [7], which is the DL providing the formal foundations for OWL 2 QL, the W3C standard ontology language for OBDA [18]. A *DL-Lite*$_\mathcal{R}$ *TBox* $\mathcal{T}$ is a finite set of axioms of the form $C \sqsubseteq D$ or $P \sqsubseteq R$, where C, D are *DL-Lite*$_\mathcal{R}$ *concepts* and P, R are *roles*, following the *DL-Lite*$_\mathcal{R}$ grammar. A *DL-Lite*$_\mathcal{R}$ *ABox* $\mathcal{A}$ is a finite set of assertions of the form $A(a)$, $P(a, b)$, where A is a concept name, P a role name, and a, b *individuals*. We call the pair $\emptyset = (\mathcal{T}, \mathcal{A})$ a *DL-Lite*$_\mathcal{R}$ *ontology*.

We consider here *first-order (FO) queries* [1], and we use $q^\mathcal{D}$ to denote the evaluation of a query q over a database $\mathcal{D}$. We use the notation $q^\mathcal{A}$ also for the evaluation of q over the ABox $\mathcal{A}$, viewed as a database. For an ontology $\emptyset$, we use $cert(q, \emptyset)$ to denote the *certain answers* of q over $\emptyset$, which are defined as the set of tuples $\mathbf{a}$ of individuals such that $\emptyset \models q(\mathbf{a})$ (where $\models$ denotes the *DL-Lite*$_\mathcal{R}$ entailment relation). We consider also various fragments of FO queries, notably *conjunctive queries* (CQs), *unions of CQs* (UCQs), and *joins of UCQs* (JUCQs) [1].

Mappings specify how to populate the concepts and roles of the ontology from the data in the underlying relational database instance of Σ. A *mapping* m is an expression of the form $L(\mathbf{f}(\mathbf{x})) \rightsquigarrow q_m(\mathbf{x})$: the *target part* $L(\mathbf{f}(\mathbf{x}))$ of m (denoted as $target(m)$) is an atom over function symbols[1] $\mathbf{f}$ and variables $\mathbf{x}$ whose predicate name L is a concept or role name; the *source part* $q_m(\mathbf{x})$ of m (denoted as $source(m)$) is a FO query over Σ with output variables[2] $\mathbf{x}$. We say that the *signature* $sign(m)$ of m is the pair $(L, \mathbf{f})$, and that m *defines* L. We also define $sign(\mathcal{M}) = \{sign(m) \mid m \in \mathcal{M}\}$.

Following [10], we split each mapping $m = L(\mathbf{f}(\mathbf{x})) \rightsquigarrow q_m(\mathbf{x})$ in $\mathcal{M}$ into two parts by introducing an intermediate view name V_m for the FO query $q_m(\mathbf{x})$. We obtain a *low-level* mapping of the form $V_m(\mathbf{x}) \rightsquigarrow q_m(\mathbf{x})$, and a *high-level* mapping of the form $L(\mathbf{f}(\mathbf{x})) \rightsquigarrow V_m(\mathbf{x})$. In the following, we abstract away the low-level mapping parts, and we consider $\mathcal{M}$ as consisting directly of the high-level mappings. In other words, we directly consider the intermediate view atoms V_m as the source part, with the semantics $V_m^\mathcal{D} = q_m^\mathcal{D}$, for each database instance $\mathcal{D}$. We denote by $\Sigma_\mathcal{M}$ the *virtual schema* consisting of the relation schemas whose names are the intermediate view symbols V_m, with attributes given by the answer variables of the corresponding source queries.

[1] For conciseness, we use here abstract function symbols in the mapping target. We remind that in concrete mapping languages, such as R2RML [8], such function symbols correspond to IRI templates used to generate object IRIs from database values.

[2] In general, the output variables of the source query might be a superset of the variables in the target, but for our purposes we can assume that they coincide.

From now on we fix an OBDA specification $\mathcal{S} = (\mathcal{T}, \mathcal{M}, \Sigma)$. Given a database instance $\mathcal{D}$ for Σ, we call the pair $(\mathcal{S}, \mathcal{D})$ an *OBDA instance*. We call the set of assertions $\mathcal{A}_{(\mathcal{M},\mathcal{D})} = \{L(\mathbf{f}(\mathbf{a})) \mid L(\mathbf{f}(\mathbf{x})) \rightsquigarrow V(\mathbf{x}) \in \mathcal{M} \text{ and } \mathbf{a} \in V(\mathbf{x})^{\mathcal{D}}\}$ the *virtual ABox exposed by $\mathcal{D}$ through $\mathcal{M}$*. Intuitively, such an ABox is obtained by evaluating, for each (high level) mapping m, its source view $V(\mathbf{x})$ over the database $\mathcal{D}$, and by using the returned tuples to instantiate the concept or role L in the target part of m. The *certain answers* $cert(q, (\mathcal{S}, \mathcal{D}))$ to a query q over an OBDA instance $(\mathcal{S}, \mathcal{D})$ are defined as $cert(q, (\mathcal{T}, \mathcal{A}_{(\mathcal{M},\mathcal{D})}))$.

In the virtual approach to OBDA, such answers are computed without actually materializing $\mathcal{A}_{(\mathcal{M},\mathcal{D})}$, by transforming the query q into a FO query q_{fo} formulated over the database schema Σ such that $q_{fo}^{\mathcal{D}'} = cert(q, (\mathcal{S}, \mathcal{D}'))$, for every OBDA instance $(\mathcal{S}, \mathcal{D}')$. To define the query q_{fo}, we introduce the following notions:

- A query q_r is a *perfect rewriting* of a query q' with respect to a TBox $\mathcal{T}$, if $cert(q', (\mathcal{T}, \mathcal{A})) = q_r^{\mathcal{A}}$, for every ABox $\mathcal{A}$ [7].
- A query q_t is an *$\mathcal{M}$-translation* of a query q', if $q_t^{\mathcal{D}} = q'^{\mathcal{A}_{(\mathcal{M},\mathcal{D})}}$, for every database $\mathcal{D}$ for Σ [19].

Notice that, by definition, all perfect rewritings (resp., translations) of q' with respect to $\mathcal{T}$ (resp., $\mathcal{M}$) are equivalent. Consider now a perfect rewriting $q_{\mathcal{T}}$ of q with respect to $\mathcal{T}$, and then a translation $q_{\mathcal{T},\mathcal{M}}$ of $q_{\mathcal{T}}$ with respect to $\mathcal{M}$. It is possible to show that such a $q_{\mathcal{T},\mathcal{M}}$ satisfies the condition stated above for q_{fo}.

Many different algorithms have been proposed for computing perfect rewritings of UCQs with respect to *DL-Lite*$_{\mathcal{R}}$ TBoxes, see, e.g., [7, 14]. As for the translation, [19] proposes an algorithm that is based on non-recursive *Datalog* [1], extended with function symbols in the head of rules, with the additional restriction that such rules never produce nested terms. We consider Datalog queries of the form (G, Π), where G is the answer atom, and Π is a set of Datalog rules following the restriction above. We abbreviate a Datalog query of the form $(q(\mathbf{x}), \{q(\mathbf{x}) \leftarrow B_1, \ldots, B_n\})$, corresponding to a CQ (possibly with function symbols), as $q(\mathbf{x}) \leftarrow B_1, \ldots, B_n$, and we also call it q. From now on, given a Datalog rule $r = A \leftarrow B_1, \ldots, B_n$, we denote by $head(r)$ its head A.

Definition 2.1 (**Unfolding of a UCQ** [19])**.** *Let $q(\mathbf{x}) \leftarrow L_1(\mathbf{v}_1), \ldots, L_n(\mathbf{v}_n)$ be a CQ. Then, the* unfolding $unf(q, \mathcal{M})$ *of q w.r.t. $\mathcal{M}$ is the Datalog query $(q_{unf}(\mathbf{x}), \Pi)$, where Π is a (up to variable renaming) minimal set of rules having the following property:*

If $((m_1, \ldots, m_n), \sigma)$ is a pair such that $\{m_1, \ldots, m_n\} \subseteq \mathcal{M}$, and
- *$m_i = L_i(\mathbf{f}_i(\mathbf{x}_i)) \rightsquigarrow V_i(\mathbf{z}_i)$, for each $1 \leq i \leq n$, and*
- *σ is a most general unifier for the set of pairs $\{(L_i(\mathbf{v}_i), L_i(\mathbf{f}_i(\mathbf{x}_i))) \mid 1 \leq i \leq n\}$,*

then the query $q_{unf}(\sigma(\mathbf{x})) \leftarrow V_1(\sigma(\mathbf{z}_1)), \ldots, V_n(\sigma(\mathbf{z}_n))$ belongs to Π.

The unfolding of a UCQ *q is the union of the unfoldings of each CQ in q.*

It has been proved in [19] that, for a UCQ q, $unf(q, \mathcal{M})$ is an $\mathcal{M}$-translation.

Theorem 2.2 (**An Unfolding is a Translation** [19])**.** *Let q be a UCQ, and $\mathcal{M}$ a set of mappings. Then,*

$$unf(q, \mathcal{M})^{\mathcal{D}} = q^{\mathcal{A}_{(\mathcal{M},\mathcal{D})}}.$$

3 Background on Cost-Based Optimization of Query Rewriting

Bursztyn et. al. [3, 4, 5] study a space of alternatives to UCQ perfect rewritings, in a setting without mappings and in which the ABox of the ontology is directly stored in the database, under suitable assumptions on the form of the chosen relations and indexes on them. Their works provides empirical evidence supporting the hypothesis that UCQ perfect rewritings could be evaluated less efficiently than alternatives forms of rewritings. In particular, they explore a space of alternatives to the traditional UCQ perfect rewriting in the form of *join of unions of conjunctive queries (JUCQs)*, and provide an algorithm that, given a cost model, selects the best choice.

Example 3.1. Consider the TBox $\mathcal{T} = \{C \sqsubseteq D\}$, and the query $q(x, y) \leftarrow D(x), P(x, y)$. A UCQ perfect rewriting of q with respect to $\mathcal{T}$ is the Datalog query $(q_{rew}(x, y), \Pi)$, where

$$\Pi = \left\{ \begin{array}{lcl} q_{rew}(x, y) & \leftarrow & C(x), P(x, y) \\ q_{rew}(x, y) & \leftarrow & D(x), P(x, y) \end{array} \right\}$$

An alternative perfect rewriting can be obtained by applying well-known distributive rules for disjunction and conjunction over q_{rew}. By doing so, we obtain the alternative perfect rewriting $(q'_{rew}(x, y), \Pi')$, where

$$\Pi' = \left\{ \begin{array}{lcl} q'_{rew}(x, y) & \leftarrow & q_{rew_D}(x), P(x, y) \\ q_{rew_D}(x) & \leftarrow & C(x) \\ q_{rew_D}(x) & \leftarrow & D(x) \end{array} \right\}$$

Observe that the above query is a JUCQ. In particular, it corresponds to the query

$$q_{cover}(x, y) \leftarrow q_{rew_D}(x), q_{rew_P}(x, y)$$

where

- $q_{rew_D}(x)$ is a UCQ perfect rewriting of $q_D(x) \leftarrow D(x)$ with respect to $\mathcal{T}$, and
- $q_{rew_P}(x, y)$ is a UCQ perfect rewriting of $q_P(x, y) \leftarrow P(x, y)$ with respect to $\mathcal{T}$. ■

In this work we study the problem of alternative translations in a full-fledged OBDA scenario with mapppings. A classical pitfall in such scenario is that it is very easy to lose the ability of performing joins over database values, so that one has to perform them over URIs constructed by applying mapping definitions. The next example shows this issue.

Example 3.2. Consider the ontology and queries from Example 3.1 and the following set of mappings:

$$\mathcal{M} = \left\{ \begin{array}{ll} C(f(a)) & \rightsquigarrow V_1(a) \\ D(f(b)) & \rightsquigarrow V_2(b) \\ D(g(c)) & \rightsquigarrow V_3(c) \\ P(f(d), h(e)) & \rightsquigarrow V_4(d, e) \end{array} \right\}$$

By applying the unfolding procedure to queries q_{rew_D} and q_{rew_P}, and joining the resulting queries, we obtain the following $\mathcal{M}$-translation q_{trans} of q'_{rew}:

$$q_{trans}(x, y) \leftarrow q_{unf_D}(x), q_{unf_P}(x, y)$$

where $q_{unf_D}(x)$ is a Datalog query of program

$$\Pi_{unf_D} = \left\{ \begin{array}{ll} q_{unf_D}(f(a)) & \leftarrow V_1(a) \\ q_{unf_D}(f(b)) & \leftarrow V_2(b) \\ q_{unf_D}(g(c)) & \leftarrow V_3(c) \end{array} \right\}$$

and $q_{unf_P}(x, y)$ is a Datalog query of program

$$\Pi_{unf_P} = \left\{ q_{unf_P}(f(d), h(e)) \leftarrow V_4(d, e) \right\}$$

The SQL query corresponding to q_{trans}, in algebra notation, is a query of the form

$$\pi_{x,y}(\pi_{x/f(a)}(V_1) \cup \pi_{x/f(b)}(V_2) \cup \pi_{x/g(c)}(V_3)) \bowtie \pi_{x/f(d),y/h(e)}(V_4)$$

where each expression of the form $x/f(\mathbf{y})$ appearing in the projections denotes the application of the function symbol f over database values instantiating the attributes in $\mathbf{y}$, and such applications construct individuals under the answer variable x. Apart from that π behaves as the standard projection operator.

Observe that q_{trans} is a JUCQ, and that it contains a join over the result of the application of function symbols to database values. For instance, in the Ontop OBDA system such application leads to an inefficient join over columns whose values are constructed from the concatenation of URI templates and database values. ■

To formalize the intuitions given in the example above, we now introduce some terminology from [3], which we use in our technical development.

Definition 3.3 (Cover [3])**.** *Let q be a query consisting of atoms $F = \{L_1, \ldots, L_n\}$. A* cover for *q is a collection C of non-empty subsets of F, called* fragments*, such that (i) $\bigcup_{f \in C} f = F$ and (ii) no fragment is included into another one.*

Definition 3.4 (Fragment Query [3])**.** *Let C be a cover for a query q. The* fragment query *$q_{|_f}(\mathbf{x}_f)$, for $f \in C$, is the query whose body consists of the atoms in f and whose answer variables $\mathbf{x}_f$ are given by the answer variables $\mathbf{x}$ of q that appear in the atoms of f, union the existential variables in f that are shared with another fragment $f' \in C$, with $f' \neq f$.*

Given a fragment query $q_{|_f}(\mathbf{x}_f)$, and a TBox $\mathcal{T}$, we use the notation $q_{|_f}^{ucq(\mathcal{T})}(\mathbf{x}_f)$ to denote a UCQ perfect rewriting of $q_{|_f}(\mathbf{x}_f)$ with respect to $\mathcal{T}$.

Definition 3.5 (Cover-based Perfect Rewriting [3])**.** *Let C be a cover for a query q, and $\mathcal{T}$ a TBox. Consider the query $q_C(\mathbf{x}) \leftarrow \bigwedge_{f \in C} q_{|_f}^{ucq(\mathcal{T})}(\mathbf{x}_f)$. Then q_C* is a cover-based JUCQ perfect rewriting of q with respect to $\mathcal{T}$ and C *if it is a perfect rewriting of q with respect to $\mathcal{T}$.*

In *DL-Lite*$_\mathcal{R}$, not every cover leads to a cover-based JUCQ perfect rewriting. Authors in [3] gave a sufficient condition, called "safety", to guarantee that every cover leads to a cover-based JUCQ perfect rewriting.

Theorem 3.6 (Cover-based query answering [3])**.** *Applying Definition 3.5 on a safe cover C for q with respect to a TBox $\mathcal{T}$, and any rewriting technique producing a UCQ, yields a cover-based JUCQ perfect rewriting q_r of q with respect to $\mathcal{T}$.*

4 Cover-based Translation in OBDA

In this section, we propose a cover-based translation of UCQ perfect rewritings. To do so, we rely on our fixed OBDA specification $\mathcal{S} = (\mathcal{T}, \mathcal{M}, \Sigma)$. In addition, we also fix a query $q(\mathbf{x})$ and a (safe) cover C for it, as well as its cover-based JUCQ perfect rewriting $q_C(\mathbf{x}) \leftarrow \bigwedge_{f \in C} q_{|_f}^{ucq(\mathcal{T})}$ with respect to $\mathcal{T}$ and C. We introduce two different characterizations of unfoldings of q_C, which produce $\mathcal{M}$-translations of q. The first characterization relies on the intuition of joining the unfoldings of each fragment query in q_C.

Definition 4.1 (Unfolding of a JUCQ – Type 1)**.** *Let C be a cover for a query q. For each $f \in C$, let Aux_f be an auxiliary predicate for $q_{|_f}^{ucq(\mathcal{T})}(\mathbf{x}_f)$, and U_f a view symbol for the unfolding $unf(q_{|_f}^{ucq(\mathcal{T})}(\mathbf{x}_f), \mathcal{M})$. Consider the set $\mathcal{M}^{aux} = \{Aux_f(\mathbf{x}_f) \rightsquigarrow U_f(\mathbf{x}_f) \mid f \in C\}$ of mappings associating the auxiliary predicates to the auxiliary view names. Then, we define the* unfolding $unf(q_C, \mathcal{M})$ of q_C with respect to $\mathcal{M}$ *as $unf(q_C^{aux}(\mathbf{x}) \leftarrow \bigwedge_{f \in C} Aux_f(\mathbf{x}_f), \mathcal{M}^{aux})$.*

Theorem 4.2 (Translation 1)**.** *Let $q_C(\mathbf{x}) \leftarrow \bigwedge_{i=1}^{n} q_{|f_i}^{ucq}$ be a JUCQ cover-based perfect rewriting. Then, the query $unf(q_C, \mathcal{M})$ is an $\mathcal{M}$-translation of q_C.*

Proof. By definition of $\mathcal{M}$-translation, we need to prove that, for each OBDA instance $\mathcal{D}$ of $\mathcal{S}$, the following equality holds:

$$unf(q_C, \mathcal{M})^{\mathcal{D}} = q_C^{\mathcal{A}_{(\mathcal{M},\mathcal{D})}}.$$

By Definition 4.1, we know that:

$$unf(q_C, \mathcal{M}) = unf(q_C^{aux}(\mathbf{x}) \leftarrow \bigwedge_{f \in C} Aux_f(\mathbf{x}_f), \mathcal{M}^{aux}), \tag{1}$$

with $\mathcal{M}^{aux}$ and Aux_f as in Definition 4.1. Let $\mathcal{D}$ be an arbitrary database instance for Σ. Then, it is not hard to verify that $q_C(\mathbf{x})^{\mathcal{A}_{(\mathcal{M},\mathcal{D})}} = q_C^{aux}(\mathbf{x})^{\mathcal{A}_{(\mathcal{M}^{aux},\mathcal{D})}}$, by applying Definition 2.1 and using the distributive property of the join operation over the union operation. By using Theorem 2.2, it holds that $q_C^{aux}(\mathbf{x})^{\mathcal{A}_{(\mathcal{M}^{aux},\mathcal{D})}} = unf(q_C^{aux}, \mathcal{M}^{aux})^{\mathcal{D}}$. By applying the transitive property of $=$, we obtain that $q_C(\mathbf{x})^{\mathcal{A}_{(\mathcal{M},\mathcal{D})}} = unf(q_C^{aux}(\mathbf{x}), \mathcal{M}^{aux})^{\mathcal{D}}$. The thesis follows by applying Equation (1), and from the fact that the choice of $\mathcal{D}$ was arbitrary. □

The above unfolding characterization for JUCQs corresponds to a translation containing SQL joins over URIs resulting from the application of function symbols to database values, in a similar fashion as in Example 3.2, rather than over (indexed) database values directly. In general, such joins cannot be evaluated efficiently by RDBMSs [22].

In the remainder of this section we propose a solution to this issue by defining an alternative characterization for the unfolding of a JUCQ, that ensures that joins are always performed on database values. To do so, we first need to introduce a number of auxiliary notions and lemmas.

Definition 4.3 (Equivalent Sets of Mappings)**.** *Two sets of mapping assertions $\mathcal{M}$ and $\mathcal{M}'$ are said to be* equivalent*, denoted as $\mathcal{M} \equiv \mathcal{M}'$, if*

For each database instance $\mathcal{D}$ of Σ, it holds that $\mathcal{A}_{(\mathcal{M},\mathcal{D})} = \mathcal{A}_{(\mathcal{M}',\mathcal{D})}$

A direct consequence of Theorem 2.2 is that unfoldings with respect to equivalent mappings must be equivalent.

Lemma 4.4. *Let $\mathcal{M}'$ be a set of mappings such that $\mathcal{M}' \equiv \mathcal{M}$. Then, for every UCQ q, it holds*

$$unf(q, \mathcal{M}) \equiv unf(q, \mathcal{M}')$$

Proof. We prove the contrapositive of the statement.
Without loss of generality, let a be such that $a \in unf(q, \mathcal{M})^{\mathcal{D}}$, and $a \notin unf(q, \mathcal{M}')^{\mathcal{D}}$, for some data instance $\mathcal{D}$ of Σ. For Theorem 2.2, the above implies that

$$a \in q^{\mathcal{A}_{(\mathcal{M},\mathcal{D})}}, a \notin q^{\mathcal{A}_{(\mathcal{M}',\mathcal{D})}}.$$

Hence, $\mathcal{A}_{(\mathcal{M},\mathcal{D})} \neq A_{(\mathcal{M}',\mathcal{D})}$, and therefore $\mathcal{M} \not\equiv \mathcal{M}'$. □

Definition 4.5 (Restriction of a Mapping to a Signature)**.** *Let $(L, \mathbf{f}) \in sign(\mathcal{M})$ be a signature in $\mathcal{M}$. Then, the* restriction $\mathcal{M}|_{(L,\mathbf{f})}$ of $\mathcal{M}$ with respect to the signature $(L, \mathbf{f})$ *is the set $\{m \in \mathcal{M} \mid sign(m) = (L, \mathbf{f})\}$ of mappings.*

Definition 4.6 (Wrap of a Mapping)**.** *Let the set $\{L(\mathbf{f}(\mathbf{v}_i)) \rightsquigarrow V_i(\mathbf{v}_i) \mid 1 \leq i \leq n\}$ be the restriction $\mathcal{M}|_{(L,\mathbf{f})}$ of $\mathcal{M}$ with respect to the signature $(L, \mathbf{f})$, and $\mathbf{f}(\mathbf{v})$ a tuple of terms over fresh variables $\mathbf{v}$. Then, the* wrap of $\mathcal{M}|_{(L,\mathbf{f})}$ *is the (singleton) set $wrap(\mathcal{M}|_{(L,\mathbf{f})}) = \{L(\mathbf{f}(\mathbf{v})) \rightsquigarrow W(\mathbf{v})\}$ of mappings, where W is a fresh view name for the Datalog query $(W(\mathbf{v}), \{W(\mathbf{v}_i) \leftarrow V_i(\mathbf{v}_i) \mid 1 \leq i \leq n\})$ with answer variables $\mathbf{v}$.*

The wrap of $\mathcal{M}$ *is the set $wrap(\mathcal{M}) = \bigcup_{(L,\mathbf{f}) \in sign(\mathcal{M})} wrap(\mathcal{M}|_{(L,\mathbf{f})})$ of mappings.*

Example 4.7. Consider the following set of mapping assertions

$$\mathcal{M}|_{(A,f)} = \left\{ \begin{array}{ll} A(f(x,y)) & \rightsquigarrow V_1(x,y) \\ A(f(a,b)) & \rightsquigarrow V_2(a,b) \end{array} \right\}$$

Let v, v' be fresh variables. Then, $wrap(\mathcal{M}|_{(A,f)}) = \{A(f(v,v')) \rightsquigarrow W(v,v')\}$, where $W(v,v')$ is the Datalog query

$$(W(v,v'), \{W(x,y) \leftarrow V_1(x,y), W(a,b) \leftarrow V_2(a,b)\})$$

Lemma 4.8. *Given a signature $(L, \mathbf{f}) \in sign(\mathcal{M})$, the following equivalence holds:*

$$\mathcal{M}|_{(L,\mathbf{f})} \equiv wrap(\mathcal{M}|_{(L,\mathbf{f})})$$

Proof. Let $\mathcal{M}|_{(L,\mathbf{f})}$ be the set $\{L(\mathbf{f}(\mathbf{v}_i)) \rightsquigarrow V_i(\mathbf{v}_i) \mid i = 1, \ldots, n\}$. Then, for every data instance $\mathcal{D}$, the virtual ABox $\mathcal{A}_{(\mathcal{M}|_{(L,\mathbf{f})},\mathcal{D})}$ can be written as:

$$\left\{ L(\mathbf{f}(\mathbf{o})) \;\middle|\; \exists i \in \{1, \ldots, n\} : \mathbf{o} \in V_i(\mathbf{v}_i)^{\mathcal{D}} \right\}. \tag{2}$$

The Set (2) can equivalently be rewritten as:

$$\left\{ L(\mathbf{f}(\mathbf{o})) \;\middle|\; \mathbf{o} \in (W(\mathbf{v}), \{W(\mathbf{v}_i) \leftarrow V_i(\mathbf{v}_i) \mid i = 1, \ldots, n\})^{\mathcal{D}} \right\}, \tag{3}$$

for some fresh view symbol W. The thesis follows by observing that the Set (3) corresponds to the definition of the virtual ABox $\mathcal{A}_{(wrap(\mathcal{M}|_{(L,\mathbf{f})}),\mathcal{D})}$. □

Lemma 4.9. *Let $\mathcal{M}$ be a set of mappings. Then $\mathcal{M} \equiv wrap(\mathcal{M})$.*

Proof. It follows directly from Lemma 4.8, and from the fact that fresh view symbols are introduced for each signature. □

The wrap operation produces a set of mappings in which there do not exist two mappings sharing the same signature. Hence, it groups the mappings for a signature into a single mapping.The next lemma formalizes this property.

Lemma 4.10. *Consider the wrap $wrap(\mathcal{M})$ of the set $\mathcal{M}$ of mappings. Then, for every mapping $m, m' \in wrap(\mathcal{M})$, it holds:*

If $sign(m) = sign(m')$, then $m = m'$

Proof. It follows directly from the definition of the *wrap* operation. □

We now introduce an operation that *splits* a mapping according to the function symbols adopted on its source part.

Definition 4.11 (Split)**.** *Let $m = L(\mathbf{x}) \rightsquigarrow U(\mathbf{x})$ be a mapping where U is a view name for a NRLP query $(U(\mathbf{x}), \{U(\mathbf{f}_i(\mathbf{x}_i)) \leftarrow V_i(\mathbf{x}_i) \mid 1 \leq i \leq n\})$. Then, the* split of m *is the set $split(m) = \{L(\mathbf{f}_i(\mathbf{x}_i)) \rightsquigarrow V_i(\mathbf{x}_i) \mid 1 \leq i \leq n\}$ of mappings. For a mapping m' not of the form above, the split of m' is defined as the set $\{m'\}$, that is, $split(m') = \{m'\}$. We denote by $split(\mathcal{M})$ the* split *of the set $\mathcal{M}$ of mappings, which is defined as the set $\bigcup_{m \in \mathcal{M}} split(m)$ of mappings.*

Example 4.12. Consider a mapping $m = A(x) \rightsquigarrow U(x)$, where $U(x) = (U(x), \Pi)$ and

$$\Pi = \left\{ \begin{array}{ll} U(h(a,b)) & \leftarrow V_1(a,b) \\ U(h(c,d)) & \leftarrow V_2(c,d) \\ U(i(e,f,g)) & \leftarrow V_3(e,f,g). \end{array} \right\}$$

Then,

$$split(\mathcal{M}) = \left\{ \begin{array}{ll} A(h(a,b)) & \rightsquigarrow V_1(a,b) \\ A(h(c,d)) & \rightsquigarrow V_2(c,d) \\ A(i(e,f,g)) & \rightsquigarrow V_3(e,f,g). \end{array} \right\}$$

Lemma 4.13. *Let m be a mapping. Then $\{m\} \equiv split(\{m\})$.*

Proof. The thesis trivially holds if m is of a form such that $\{m\} = split(m)$. If $\{m\} \neq split(m)$, then m must be of the form $L(\mathbf{x}) \rightsquigarrow U(\mathbf{x})$, in which U is a view name for a NRLP query $(U(\mathbf{x}), \Pi)$, where $\Pi = \{U(\mathbf{f}_i(\mathbf{x}_i)) \leftarrow V_i(\mathbf{x}_i) \mid 1 \leq i \leq n\}$. By Definition 4.3, we need to prove that, for each instance $\mathcal{D}$ of $\mathcal{S}$, $\mathcal{A}_{(\{m\},\mathcal{D})} = \mathcal{A}_{(split(\{m\}),\mathcal{D})}$. Let $\mathcal{D}$ be an arbitrary database instance. By definition of virtual ABox:

$$A_{(\{m\},\mathcal{D})} = \left\{L(\mathbf{f}_i(\mathbf{a}_i)) \;\middle|\; \mathbf{f}_i(\mathbf{a}_i) \in U(\mathbf{x})^{\mathcal{D}}\right\}$$

The set above is equal to

$$\begin{aligned}&\left\{L(\mathbf{f}_i(\mathbf{a}_i)) \;\middle|\; \exists(q(\mathbf{f}_i(\mathbf{x}_i)) \leftarrow V_i(\mathbf{x}_i)) \in \Pi \text{ s.t. } \mathbf{a}_i \in V_i(\mathbf{x}_i)^{\mathcal{D}}\right\}\\ =&\left\{L(\mathbf{f}_i(\mathbf{a}_i)) \;\middle|\; \exists(L(\mathbf{f}_i(\mathbf{x}_i)) \rightsquigarrow V_i(\mathbf{x}_i)) \in \text{split}(\{m\}) \text{ s.t. } \mathbf{a}_i \in V_i(\mathbf{x}_i)^{\mathcal{D}}\right\} = \mathcal{A}_{(split(\{m\}),\mathcal{D})}.\end{aligned}$$

□

Corollary 4.14. *Let $\mathcal{M}$ be a set of mappings. Then $\mathcal{M} \equiv split(\mathcal{M})$.*

Proof. By Definition 4.3, we need to prove that, for each instance $\mathcal{D}$ of $\mathcal{S}$, $\mathcal{A}_{(\mathcal{M},\mathcal{D})} = \mathcal{A}_{(split(\mathcal{M}),\mathcal{D})}$. Let $\mathcal{D}$ be a data instance for $\mathcal{S}$. The following trivially holds:

$$\mathcal{A}_{(\mathcal{M},\mathcal{D})} = \bigcup_{m \in \mathcal{M}} \mathcal{A}_{(\{m\},\mathcal{D})}.$$

By Lemma 4.14,

$$\bigcup_{m \in \mathcal{M}} \mathcal{A}_{(\{m\},\mathcal{D})} = \bigcup_{m \in \mathcal{M}} \mathcal{A}_{(split(\{m\}),\mathcal{D})}.$$

By Definition 4.11, we obtain

$$\bigcup_{m \in \mathcal{M}} \mathcal{A}_{(split(\{m\}),\mathcal{D})} = \mathcal{A}_{(split(\mathcal{M}),\mathcal{D})}.$$

□

Definition 4.15 (Unfolding of a JUCQ – Type 2)**.** *Let q_C^{aux} be a query and $\mathcal{M}^{aux}$ a set of mappings as in Definition 4.1. Then, the* optimized unfolding *$unf_{opt}(q_C(\mathbf{x}), \mathcal{M})$ of q_C with respect to $\mathcal{M}$ is defined as $unf(q_C^{aux}(\mathbf{x}), wrap(split(\mathcal{M}^{aux})))$.*

Observe that the optimized unfolding of a JUCQ is a *union of JUCQs* (UJUCQ). Moreover, where each JUCQ produces answers built from a *single* tuple of function symbols, if all the attributes are kept in the answer. The next example, aimed at clarifying the notions introduced so far, illustrates this.

Example 4.16. Let $q(x, y, z) \leftarrow P_1(x, y), C(x), P_2(x, z)$, and consider a cover $\{f_1, f_2\}$ generating fragment queries $q|_{f_1} = q(x, y) \leftarrow P_1(x, y), C(x)$ and $q|_{f_2} = q(x, z) \leftarrow P_2(x, z)$. Consider the set of mappings

$$\mathcal{M} = \left\{ \begin{array}{llll} P_1(f(a), g(b)) \rightsquigarrow V_1(a, b) & & P_1(f(a), g(b)) & \rightsquigarrow V_2(a, b) \\ P_1(h(a), i(b)) \rightsquigarrow V_3(a, b) & & C(f(a)) & \rightsquigarrow V_4(a) \\ P_2(f(a), k(b)) \rightsquigarrow V_5(a, b) & & P_2(f(a), h(b)) & \rightsquigarrow V_6(a, b) \end{array} \right\}$$

Translation I. According to Definition 4.1, the JUCQ $q(x, y, z) \leftarrow q|_{f_1}(x, y), q|_{f_2}(x, z)$ can be rewritten as the auxiliary query $q^{aux}(x, y, z) = Aux_1(x, y), Aux_2(x, z)$ over mappings

$$\mathcal{M}^{aux} = \quad \left\{ \quad Aux_1(x, y) \rightsquigarrow U_1(x, y) \qquad Aux_2(x, z) \rightsquigarrow U_2(x, z) \quad \right\}$$

where U_1 is a view name for $unf(q|_{f_1}(x, y), \mathcal{M}) = (U_1(x, y), \Pi_1)$, and U_2 is a view name for $unf(q|_{f_2}(x, z), \mathcal{M}) = (U_2(x, z), \Pi_2)$, such that

$$\Pi_1 = \left\{ \begin{array}{ll} U_1(f(a), g(b)) & \leftarrow V_1(a, b), V_4(a) \\ U_1(f(a), g(b)) & \leftarrow V_2(a, b), V_4(a) \end{array} \right\} \qquad \Pi_2 = \left\{ \begin{array}{ll} U_2(f(a), k(b)) & \leftarrow V_5(a, b) \\ U_2(f(a), h(b)) & \leftarrow V_6(a, b) \end{array} \right\}$$

Translation II. By Definition 4.11, we compute the split of $\mathcal{M}^{aux}$:

$$split(\mathcal{M}^{aux}) = \left\{ \begin{array}{ll} Aux_1(f(a), g(b)) \rightsquigarrow V_1(a, b), V_4(a) & Aux_2(f(a), k(b)) \rightsquigarrow V_5(a, b) \\ Aux_1(f(a), g(b)) \rightsquigarrow V_2(a, b), V_4(a) & Aux_2(f(a), h(b)) \rightsquigarrow V_6(a, b) \end{array} \right\}$$

By Definition 4.6, we compute the wrap of $split(\mathcal{M}^{aux})$:

$$wrap(split(\mathcal{M}^{aux})) = \left\{ \begin{array}{ll} Aux_1(f(a), g(b)) \rightsquigarrow W_3(a, b) & Aux_2(f(a), k(b)) \rightsquigarrow W_4(a, b) \\ & Aux_2(f(a), h(b)) \rightsquigarrow W_5(a, b) \end{array} \right\}$$

where $W_3(a,b)$, $W_4(a,b)$, $W_5(a,b)$ are Datalog queries whose programs are respectively

$$\Pi_3 = \left\{ \begin{array}{ll} W_3(a,b) & \leftarrow V_1(a,b), V_4(a) \\ W_3(a,b) & \leftarrow V_2(a,b), V_4(a) \end{array} \right\} \qquad \Pi_4 = \left\{ W_4(a,b) \leftarrow V_5(a,b) \right\} \qquad \Pi_5 = \left\{ W_5(a,b) \leftarrow V_6(a,b) \right\}$$

Finally, by Definition 4.15 we compute the optimized unfolding of q_C with respect to $\mathcal{M}$:

$$unf_{opt}(q_C(x,y,z), \mathcal{M}) = unf(q^{aux}(x,y,z), wrap(split(\mathcal{M}^{aux}))) = (q^{aux}_{unf}(x,y,z), \Pi_{unf})$$

where

$$\Pi_{unf} = \left\{ \begin{array}{l} q^{aux}_{unf}(f(a), g(b), k(b')) \leftarrow W_3(a,b), W_4(a,b') \\ q^{aux}_{unf}(f(a), g(b), h(b')) \leftarrow W_3(a,b), W_5(a,b') \end{array} \right\}$$

Observe that $unf_{opt}(q_C(x,y,z), \mathcal{M})$ is a UJUCQ. Moreover, each of the two JUCQs in q^{aux}_{unf} contributes with answers built out of a specific tuple of function symbols. ■

Theorem 4.17 (Translation 2)**.** *The query $unf_{opt}(q_C, \mathcal{M})$ is an $\mathcal{M}$-translation of q_C.*

Proof. By Definition 4.15,

$$unf_{opt}(q_C(\mathbf{x}), \mathcal{M}) = unf(q^{aux}_C(\mathbf{x}), wrap(split(\mathcal{M}^{aux}))).$$

By Lemma 4.9 and Corollary 4.14, $wrap(split(\mathcal{M}^{aux})) \equiv \mathcal{M}^{aux}$. Hence, by Lemma 4.4,

$$unf(q^{aux}_C(\mathbf{x}), wrap(split(\mathcal{M}^{aux}))) \equiv unf(q^{aux}_C(\mathbf{x}), \mathcal{M}^{aux}).$$

The thesis is proved by observing that, according to Theorem 4.2, $unf(q^{aux}_C(\mathbf{x}), \mathcal{M}^{aux})$ is an $\mathcal{M}$-translation of q_C. □

5 Unfolding Cardinality Estimation

For convenience, in this section, we use relational algebra notation [1] for CQs. To deal with multiple occurrences of the same predicate in a CQ, the corresponding algebra expression would contain renaming operators. However, in our cardinality estimations we need to understand when two attributes actually refer to the same relation, and this information is lost in the presence of renaming. Instead of introducing renaming, we first explicitly replace multiple occurrences of the same predicate name in the CQ by aliases (under the assumption that aliases for the same predicate name are interpreted as the same relation). Specifically, we use alias $V_{[i]}$ to represent the i-th occurrence of predicate name V in the CQ. Then, when translating the aliased CQ to algebra, we use *fully qualified attribute names* (i.e., each attribute name is prefixed with the (aliased) predicate name). So, to reconstruct the relation name V to which an attribute $V_{[i]}.x$ refers, it suffices to remove the occurrence information $_{[i]}$ from the prefix $V_{[i]}$. When the actual occurrence of V is not relevant, we use $V_{[\cdot]}$ to denote the alias.

We first consider the restricted case of CQs, which we call *basic CQs*, whose algebra expression is of the form

$$E = V^0_{[\cdot]} \bowtie_{\theta_1} V^1_{[\cdot]} \bowtie_{\theta_2} \cdots \bowtie_{\theta_n} V^n_{[\cdot]},$$

where, the V^is denote predicate names, and for each $i \in \{1, \ldots, n\}$, the join condition θ_i is a single equality, i.e., of the form $V^j_{[\cdot]}.\mathbf{x} = V^i_{[\cdot]}.\mathbf{y}$, for some $j < i$. We then extend our cardinality estimation to arbitrary CQs, allowing for projections and arbitrary joins.

Given a basic CQ E as above, we denote by $E^{(m)}$, for $1 \leq m \leq n$, the sub-expression of E up to the m-th join operator, namely $E^{(m)} = V^0_{[\cdot]} \bowtie_{\theta_1} V^1_{[\cdot]} \bowtie_{\theta_2} \cdots \bowtie_{\theta_m} V^m_{[\cdot]}$.

In the following, in addition to an OBDA specification, we also fix a database instance $\mathcal{D}$ for Σ. We use V and W to denote relation names (with an associated relation schema) in the virtual schema $\mathcal{M}_\Sigma$, whose associated relations consist of (multi)sets of labeled tuples (see the *named perspective* in [1]). Given a relation S, we denote by $|S|$ the number of (distinct) tuples in S, by $\pi_L(S)$ the *projection* of S over attributes L (under *set-semantics*), by $S|_L$ the *restriction* of S over attributes L (under *bag-semantics*), and by $\pi_{L_1}(S_1) \Cap \pi_{L_2}(S_2)$ intersection of relations disregarding attribute names, i.e., $\pi_{L_1}(S_1) \cap \rho_{L_2 \mapsto L_1}(\pi_{L_2}(S_2))$. We also use the classical notation $P(\alpha)$ to denote the probability that an event α happens.

5.1 Background on Cardinality Estimation

In this subsection we list a number of assumptions that are commonly made by models of cardinality estimations proposed in the database literature (e.g., see [24]). Some of these assumptions will be maintained also in our cardinality estimator, while others will be relaxed or dropped due to the additional information given by the structure of the mappings and the ontology, which is not available in a traditional database setting.

Uniform distribution in the interval. This assumption holds when values are uniformly distributed across one interval. Formally:

$$P(C < v) = (v - min(C))/(max(C) - min(C)) \tag{a1}$$

for each value v in a column C.

Uniform distribution across distinct values. This assumption holds when values in a column are uniformly distributed across the range of values. Formally:

$$\forall v_1, v_2 \in C.\ P(C = v_1) = P(C = v_2) \tag{a2}$$

for all values v_1 and v_2 in a column C.

A consequence of this assumption is that, for each relation name T and set of attributes $\mathbf{x}$,

$$P(T|_{\mathbf{x}} = \mathbf{v}) = P(\pi_{\mathbf{x}}(T) = \mathbf{v}) = 1/|\pi_{\mathbf{x}}(T)|, \forall \mathbf{v} \in \pi_{\mathbf{x}}(\pi_{\mathbf{x}}(T)).$$

Hence, *number of repetitions per tuple of elements* can be calculated as

$$|T| \cdot P(T|_{\mathbf{x}} = \mathbf{v}) = \frac{|T|}{|\pi_{\mathbf{x}}(T)|}. \tag{4}$$

From Equation (4), we directly compute the *ratio of distinct tuples over* $\mathbf{x}$ *in* T as:

$$\frac{|\pi_{\mathbf{x}}(T)|}{|T|}$$

Independent Distributions. This assumption holds when the distributions between different attributes are independent. Formally:

$$P(C_1 = v_1 \mid C_2 = v2) = P(C_1 = v_1) \tag{a3}$$

Facing Values. This assumption holds when attributes join "as much as possible". That is, given a join $T \bowtie_{\mathbf{x}=\mathbf{y}} S$, it is assumed that

$$|\pi_{\mathbf{x}}(T) \cap \pi_{\mathbf{y}}(S)| = min(|\pi_{\mathbf{x}}(T)|, |\pi_{\mathbf{y}}(S)|). \tag{a4}$$

Under these assumptions, the cardinality of a join $V \bowtie_{\mathbf{x}=\mathbf{y}} W$ can be estimated [25] to be:

$$fv_{\mathcal{D}}(V \bowtie_{\mathbf{x}=\mathbf{y}} W) \cdot \frac{|V^{\mathcal{D}}|}{dist_{\mathcal{D}}(V, \mathbf{x})} \cdot \frac{|W^{\mathcal{D}}|}{dist_{\mathcal{D}}(W, \mathbf{y})} \tag{5}$$

where $fv_{\mathcal{D}}(V \bowtie_{\mathbf{x}=\mathbf{y}} W)$ is an estimation of the number of distinct values satisfying the join condition $V \bowtie_{\mathbf{x}=\mathbf{y}} W$ (i.e., $fv_{\mathcal{D}}$ estimates $|\pi_{\mathbf{x}}(V^{\mathcal{D}}) \pitchfork \pi_{\mathbf{y}}(W^{\mathcal{D}})|$, and $dist_{\mathcal{D}}(V, \mathbf{x})$ (resp., $dist_{\mathcal{D}}(W, \mathbf{y})$) corresponds to the estimation of $|\pi_{\mathbf{x}}(V^{\mathcal{D}})|$ (resp., $|\pi_{\mathbf{y}}(W^{\mathcal{D}})|$), both calculated according to the aforementioned assumptions. Note that the fractions such as $\frac{|V^{\mathcal{D}}|}{dist_{\mathcal{D}}(V,\mathbf{x})}$ estimate the number of tuples associated to each value that satisfies the join condition, and derive directly from Assumption (a2).

5.2 Cardinality Estimation of CQs

Cardinality Estimator. Given a basic CQ E', $ce_{\mathcal{D}}(E')$ estimates the number $|E'^{\mathcal{D}}|$ of distinct results in the evaluation of E' over $\mathcal{D}$. We define it as

$$ce_{\mathcal{D}}(E \bowtie_{V_{[p]}.\mathbf{x}=W_{[q]}.\mathbf{y}} W_{[q]}) = \begin{cases} \left\lceil \frac{fv_{\mathcal{D}}(V_{[p]} \bowtie_{V_{[p]}.\mathbf{x}=W_{[q]}.\mathbf{y}} W_{[q]}) \cdot |V^{\mathcal{D}}| \cdot |W^{\mathcal{D}}|}{dist_{\mathcal{D}}(V, V_{[p]}.\mathbf{x}) \cdot dist_{\mathcal{D}}(W, W_{[q]}.\mathbf{y})} \right\rceil, & \text{if } E = V \\ \left\lceil \frac{fv_{\mathcal{D}}(E \bowtie_{V_{[p]}.\mathbf{x}=W_{[q]}.\mathbf{y}} W_{[q]}) \cdot ce_{\mathcal{D}}(E) \cdot |W^{\mathcal{D}}|}{dist_{\mathcal{D}}(E, V_{[p]}.\mathbf{x}) \cdot dist_{\mathcal{D}}(W, V_{[p]}.\mathbf{y})} \right\rceil, & \text{otherwise.} \end{cases} \tag{6}$$

Our cardinality estimator exploits assumptions (a2) and (a3) above, and relies on our definitions of the *facing values estimator* $fv_{\mathcal{D}}$ and of the *distinct values estimator* $\text{dist}_{\mathcal{D}}$, which are based on additional statistics collected with the help of the mappings, instead of being based on assumptions (a1) and (a4), as in Formula (5).

T_1

a	b	...
1	4	...
2	8	...
3	12	...
4	16	...
5	20	...

T_2

c	d	...
1	2	...
3	4	...
5	6	...
7	8	...
9	10	...
11	2	...
13	4	...
15	6	...
17	8	...
19	10	...

T_3

e	f	...
1	1	...
2	8	...
3	16	...
4	24	...
5	32	...
6	40	...
7	48	...
8	56	...
9	64	...
10	72	...

Figure 1: Data instance $\mathcal{D}$.

Facing Values Estimator. Given a basic CQ $E' = E \bowtie_{V_{[p]}.\mathbf{x}=W_{[q]}.\mathbf{y}} W_{[q]}$, the estimation $fv_{\mathcal{D}}(E')$ of the cardinality $|\pi_{V.\mathbf{x}}(E^{\mathcal{D}}) \Cap \pi_{W.\mathbf{y}}(W^{\mathcal{D}})|$ is defined as

$$fv_{\mathcal{D}}(E \bowtie_{V_{[p]}.\mathbf{x}=W_{[q]}.\mathbf{y}} W_{[q]}) = \begin{cases} |\pi_{\mathbf{x}}(V^{\mathcal{D}}) \Cap \pi_{\mathbf{y}}(W^{\mathcal{D}})|, & \text{if } E = V \\ \left\lceil |\pi_{\mathbf{x}}(V^{\mathcal{D}}) \Cap \pi_{\mathbf{y}}(W^{\mathcal{D}})| \cdot \dfrac{dist_{\mathcal{D}}(E, V_{[p]}.\mathbf{x})}{dist_{\mathcal{D}}(V, V_{[p]}.\mathbf{x})} \right\rceil, & \text{otherwise,} \end{cases} \tag{7}$$

where $|\pi_{\mathbf{x}}(V^{\mathcal{D}}) \Cap \pi_{\mathbf{y}}(W^{\mathcal{D}})|$ is assumed to be a statistic available after having analyzed the mappings together with the data instance. The fraction $\frac{dist_{\mathcal{D}}(E,V_{[p]}.\mathbf{x})}{dist_{\mathcal{D}}(V,V_{[p]}.\mathbf{x})}$ is a scaling factor relying on assumption (a2).

Distinct Values Estimator.

Definition 5.1 (Equivalent Attributes)**.** *Let Q be a set of qualified attributes, and E be basic CQ. We define the set $ea(E, Q)$ of equivalent attributes of Q in E as $\bigcup_{i>0} C_i$, where*

- $C_1 := \{Q\}$
- $C_{n+1} := C_n \cup \{Q' \mid \exists Q'' \in C_n \text{ s.t. } Q' = Q'' \text{ or } Q'' = Q' \text{ is a join condition in } E\}$, $n \geq 1$

Definition 5.2 (Join Sub-Expression)**.** *Given a basic CQ E and a set $V_{[p]}.\mathbf{x}$ of qualified attributes, the expression $se(E, V_{[p]}.\mathbf{x})$ denotes the longest sub-expression $E^{(n)}$ in E, for some $n > 1$, such that $E^{(n)} = E^{(n-1)} \bowtie_{W_{[q]}.\mathbf{y}=U_{[r]}.\mathbf{z}} U_{[r]}$, for some relation name W, tuples of attributes $\mathbf{y}$ and $\mathbf{z}$ such that $U_{[r]}.\mathbf{z} \in ea(E, V_{[p]}.\mathbf{x})$, if $E^{(n)}$ exists, and $\perp$ otherwise.*

For E and $V_{[p]}.\mathbf{x}$, the estimation $dist_{\mathcal{D}}(E, V_{[p]}.\mathbf{x})$ of the cardinality $|\pi_{V_{[p]}.\mathbf{x}}(E^{\mathcal{D}})|$ is defined as

$$dist_{\mathcal{D}}(E, V_{[p]}.\mathbf{x}) = \begin{cases} |\pi_{\mathbf{x}}(V^{\mathcal{D}})|, & \text{if } E = V \\ \min\left\{ \left\lceil fv_{\mathcal{D}}(E') \cdot \dfrac{ce_{\mathcal{D}}(E)}{ce_{\mathcal{D}}(E')} \right\rceil, fv_{\mathcal{D}}(E') \right\}, & \text{if } se(E, V_{[p]}.\mathbf{x}) = E' \neq \perp \\ \min\left\{ \left\lceil |\pi_{\mathbf{x}}(V^{\mathcal{D}})| \cdot \dfrac{ce_{\mathcal{D}}(E)}{|V^{\mathcal{D}}|} \right\rceil, |\pi_{\mathbf{x}}(V^{\mathcal{D}})| \right\}, & \text{otherwise.} \end{cases} \tag{8}$$

where $|\pi_{\mathbf{x}}(V^{\mathcal{D}})|$ is assumed to be a statistic available after having analyzed the mappings together with the data instance. Observe that the fractions $\frac{ce_{\mathcal{D}}(E)}{ce_{\mathcal{D}}(E')}$ and $\frac{ce_{\mathcal{D}}(E)}{|V^{\mathcal{D}}|}$ are again scaling factors relying on assumption (a2). Also, $dist_{\mathcal{D}}(E, V.\mathbf{x})$ must not increase when the number of joins in E increases, which explains the use of $\min$ for the case where the number of distinct results in E increases with the number of joins.

Example 5.3. Consider the data instance $\mathcal{D}$ from Figure 1. Relevant statistics are:

- $|T_1^{\mathcal{D}}| = 5, \quad |T_2^{\mathcal{D}}| = |T_3^{\mathcal{D}}| = 10$

- $|\pi_a(T_1^{\mathcal{D}})| = |\pi_d(T_2^{\mathcal{D}})| = 5, \quad |\pi_c(T_2^{\mathcal{D}})| = |\pi_f(T_3^{\mathcal{D}})| = |\pi_e(T_3^{\mathcal{D}})| = 10,$
- $|\pi_a(T_1^{\mathcal{D}}) \pitchfork \pi_c(T_2^{\mathcal{D}})| = 3, \quad |\pi_d(T_2^{\mathcal{D}}) \pitchfork \pi_e(T_3^{\mathcal{D}})| = 5, \quad |\pi_a(T_1^{\mathcal{D}}) \pitchfork \pi_f(T_3^{\mathcal{D}})| = 1.$

We calculate $ce_{\mathcal{D}}(E)$ for the basic CQ $E = T_1 \bowtie_{T_1.a=T_2.c} T_2 \bowtie_{T_2.d=T_3.e} T_3 \bowtie_{T_1.a=T_3'.f} T_3'$, where T_3' is an alias (written in this way for notational convenience) for the table T_3. To do so, we first need to calculate the estimations $ce_{\mathcal{D}}(E^{(1)})$ and $ce_{\mathcal{D}}(E^{(2)})$.

$$\begin{aligned} ce_{\mathcal{D}}(E^{(1)}) &= ce_{\mathcal{D}}(T_1 \bowtie_{T_1.a=T_2.c} T_2) = \left\lceil \frac{fv_{\mathcal{D}}(T_1 \bowtie_{T_1.a=T_2.c} T_2) \cdot |T_1^{\mathcal{D}}| \cdot |T_2^{\mathcal{D}}|}{dist_{\mathcal{D}}(T_1, a) \cdot dist_{\mathcal{D}}(T_2, c)} \right\rceil \\ &= \left\lceil \frac{|\pi_a(T_1^{\mathcal{D}}) \pitchfork \pi_c(T_2^{\mathcal{D}})| \cdot |T_1^{\mathcal{D}}| \cdot |T_2^{\mathcal{D}}|}{|\pi_a(T_1^{\mathcal{D}})| \cdot |\pi_c(T_2^{\mathcal{D}})|} \right\rceil = \lceil (3 \cdot 5 \cdot 10)/(5 \cdot 10) \rceil = 3 \\ ce_{\mathcal{D}}(E^{(2)}) &= ce_{\mathcal{D}}(E^{(1)} \bowtie_{T_2.d=T3.e} T_3) = \left\lceil \frac{fv_{\mathcal{D}}(E^{(1)} \bowtie_{T_2.d=T3.e} T_3) \cdot ce_{\mathcal{D}}(E^{(1)}) \cdot |T_3^{\mathcal{D}}|}{dist_{\mathcal{D}}(E^{(1)}, T_2.d) \cdot dist_{\mathcal{D}}(T_3, e)} \right\rceil \end{aligned} \tag{9}$$

By Formula (8), $dist_{\mathcal{D}}(E^{(1)}, T_2.d)$ in Formula (9) can be calculated as

$$\begin{aligned} dist_{\mathcal{D}}(E^{(1)}, T_2.d) &= \min \left\{ \left\lceil \frac{|\pi_d(T_2^{\mathcal{D}})|}{|T_2^{\mathcal{D}}|} \cdot ce_{\mathcal{D}}(E^{(1)}) \right\rceil, |\pi_d(T_2^{\mathcal{D}})| \right\} \\ &= \min \left\{ \left\lceil \frac{5}{10} \cdot 3 \right\rceil, 5 \right\} = \left\lceil \frac{3}{2} \right\rceil = 2 \end{aligned}$$

By Formula (7), $fv_{\mathcal{D}}(E^{(1)} \bowtie_{T_2.d=T3.e} T_3)$ in Formula (9) can be calculated as

$$\begin{aligned} fv_{\mathcal{D}}(E^{(1)} \bowtie_{T_2.d=T3.e} T_3) &= \left\lceil \frac{fv_{\mathcal{D}}(T_2 \bowtie_{T_2.d=T3.e} T_3)}{dist_{\mathcal{D}}(T_2, d)} \cdot dist_{\mathcal{D}}(E^{(1)}, T_2.d) \right\rceil \\ &= \left\lceil \frac{|\pi_d(T_2^{\mathcal{D}}) \pitchfork \pi_e(T_3^{\mathcal{D}})|}{|\pi_d(T_2^{\mathcal{D}})|} \cdot dist_{\mathcal{D}}(E^{(1)}, T_2.d) \right\rceil = \left\lceil \frac{5}{5} \cdot 2 \right\rceil = 2 \end{aligned}$$

By plugging the values for $fv_{\mathcal{D}}$ and $dist_{\mathcal{D}}$ in Formula (9), we obtain

$$ce_{\mathcal{D}}(E^{(2)}) = \lceil (2 \cdot 3 \cdot 10)/(2 \cdot 10) \rceil = 3$$

We are now ready to calculate the cardinality of E, which is given by the formula

$$ce_{\mathcal{D}}(E) = ce_{\mathcal{D}}(E^{(2)} \bowtie_{T_1.a=T_3'.f} T_3') = \left\lceil \frac{fv_{\mathcal{D}}(E^{(2)} \bowtie_{T_1.a=T_3'.f} T_3') \cdot ce_{\mathcal{D}}(E^{(2)}) \cdot |T_3^{\mathcal{D}}|}{dist_{\mathcal{D}}(E^{(2)}, T_1.a) \cdot dist_{\mathcal{D}}(T_3, f)} \right\rceil \tag{10}$$

By Formula (8), $dist_{\mathcal{D}}(E^{(2)}, T_1.a)$ in Formula (10) can be computed as

$$dist_{\mathcal{D}}(E^{(2)}, T_1.a) = \min \left\{ \left\lceil \frac{fv_{\mathcal{D}}(E^{(1)})}{ce_{\mathcal{D}}(E^{(1)})} \cdot ce_{\mathcal{D}}(E^{(2)}) \right\rceil, fv_{\mathcal{D}}(E^{(1)}) \right\} = \min \left\{ \left\lceil \frac{3}{3} \cdot 3 \right\rceil, 3 \right\} = 3$$

Then, by Formula (7), $fv_{\mathcal{D}}(E^{(2)} \bowtie_{T_1.a=T_3'.f} T_3')$ in Formula (10) can be computed as

$$fv_{\mathcal{D}}(E^{(2)} \bowtie_{T_1.a=T_3'.f} T_3') = \left\lceil \frac{fv_{\mathcal{D}}(T_1 \bowtie_{T_1.a=T_3'.f} T_3')}{dist_{\mathcal{D}}(T_1, a)} \cdot dist_{\mathcal{D}}(E^{(2)}, T_1.a) \right\rceil = \left\lceil \frac{3}{5} \right\rceil = 1$$

By plugging the values for $fv_{\mathcal{D}}$ and $dist_{\mathcal{D}}$ in (10), we finally obtain

$$ce_{\mathcal{D}}(E) = \lceil (1 \cdot 3 \cdot 10)/(3 \cdot 10) \rceil = 1$$

Observe that, in this example, our estimation is exact, that is, $ce_{\mathcal{D}}(E) = |E^{\mathcal{D}}|$. ■

5.3 Extending Cardinality Estimation to Non-basic CQs

We study now how to extend the cardinality estimation developed above to arbitrary join conditions and to CQs with existential variables.

5.3.1 Extending Cardinality Estimation to Arbitrary Join Conditions

To extend the cardinality estimation to the case of arbitrary join conditions, we exploit the following property of theta-joins:

$$\sigma_{\theta_2}(T_1 \bowtie_{\theta_1} T_2) = T_1 \bowtie_{\theta_1 \wedge \theta_2} T_2$$

The cardinality of a CQ without existential variables, and with n atoms and m join conditions ($m \geq n$) can be estimated in two steps. In the first step, we estimate through the equations in the previous section the cardinality of a basic CQ of n atoms over $n-1$ join conditions. In the second step, we multiply the number obtained in the first step by the probability that the additional $\theta_n, \ldots, \theta_m$ conditions are all satisfied. Such probability can be easily calculated by relying on the uniformity assumptions. We illustrate this on the case of three atoms and three join conditions. The generalization to an arbitrary number of atoms and join conditions is straightforward.

Example 5.4.

$$ce_{\mathcal{D}}((T_1 \bowtie_{T_1.a=T_2.d} T_2) \bowtie_{T_1.h=T_3.i \wedge T_2.d=T_3.e} T_3) = ce_{\mathcal{D}}((T_1 \bowtie_{T_1.a=T_2.d} T_2) \bowtie_{T_1.h=T_3.i} T_3) \cdot \frac{fv_{\mathcal{D}}(T_2 \bowtie_{T_2.d=T_3.e} T_3)}{dist_{\mathcal{D}}(T_2, T_2.d)}$$

where $\frac{fv_{\mathcal{D}}(T_2 \bowtie_{T_2.d=T_3.e} T_3)}{dist_{\mathcal{D}}(T_2, T_2.d)}$ is an estimation for $P(T_2.d = T_3.e)$ under our assumptions.

5.3.2 Extending Cardinality Estimation to CQs with Existential Variables (Projection)

Consequences of Cardinality Assumptions In this paragraph we discuss some important consequences of our assumptions of uniform distribution across distinct values (Assumption (a2)) and of independent distributions (Assumption (a3)).

Consider a set of columns $C_1, \ldots, C_n$ over a table T. Then, Assumption (a2) can be written as

$$\forall \mathbf{v} \in T|_{C_1,\ldots,C_n} : P(T|_{C_1,\ldots,C_n} = \mathbf{v}) = \frac{1}{|\pi_{C_1,\ldots,C_n} T|}. \tag{11}$$

A consequence of Assumption (a3) is that

$$\forall (v_1, \ldots, v_n) \in C_1 \times \cdots \times C_n : P(T|_{C_1} = v_1 \cap \cdots \cap T|_{C_n} = v_n) = P(T|_{C_1} = v_1) \cdot \cdots \cdot P(T|_{C_n} = v_n). \tag{12}$$

By Assumption (a2), we know that

$$\forall i \in \{1, \ldots, n\} : P(T|_{C_i} = v_i) = \frac{1}{|\pi_{C_i} T|}.$$

Hence, Equation (12) becomes

$$\forall (v_1, \ldots, v_n) \in C_1 \times \cdots \times C_n : P(T|_{C_1} = v_1 \cap \cdots \cap T|_{C_n} = v_n) = \Pi_{i=1}^{n} \frac{1}{|\pi_{C_i} T|}. \tag{13}$$

We observe that Equation (13) can equivalently be rewritten as

$$\forall \mathbf{v} \in \pi_{C_1,\ldots,C_n} T : P(\pi_{C_1,\ldots,C_n} T = \mathbf{v}) = \frac{1}{\Pi_{i=1}^{n} \frac{1}{|\pi_{C_i} T|}}. \tag{14}$$

Therefore, by Equation (11), we conclude that

$$|\pi_{C_1,\ldots,C_n} T| = \Pi_{i=1}^{n} |\pi_{C_i} T|. \tag{15}$$

Let $\pi_{V_{[\cdot]}^1.\mathbf{x}_1,\ldots,V_{[\cdot]}^n.\mathbf{x}_n}(E)$ be a CQ, where E is a CQ. Then, by Equation (15), we estimate the cardinality $|\pi_{V_{[\cdot]}^1.\mathbf{x}_1,\ldots,V_{[\cdot]}^n.\mathbf{x}_n}(E)|$ through the formula[3]

$$\min\{ce_{\mathcal{D}}(E), \Pi_{i \in \{1,\ldots,n\}} dist_{\mathcal{D}}(E, V_{[\cdot]}^i.\mathbf{x}_i)\}$$

[3]See also `http://adrem.ua.ac.be/sites/adrem.ua.ac.be/files/db2-selectivity.pdf`, page 13.

5.4 Collecting the Necessary Base Statistics

The estimators introduced above assume a number of statistics to be available. We now show how to compute such statistics on a data instance by analyzing the mappings. Consider a set $\mathcal{M} = \{L_i(\mathbf{f}_i(\mathbf{v}_i)) \rightsquigarrow V_i(\mathbf{v}_i) \mid 1 \leq i \leq n\}$ of mappings and a data instance $\mathcal{D}$. We store the statistics:

(S_1) $|V_i^{\mathcal{D}}|$, for each $i \in \{1, \ldots, n\}$;

(S_2) $|\pi_{\mathbf{x}}(V_i^{\mathcal{D}})|$, if $f(\mathbf{x})$ is a term in $\mathbf{f}_i(\mathbf{v}_i)$, for some function symbol f and $i \in \{1, \ldots, n\}$;

(S_3) $|\pi_{\mathbf{x}}(V_i^{\mathcal{D}}) \Cap \pi_{\mathbf{y}}(V_j^{\mathcal{D}})|$, if $f(\mathbf{x})$ is a term in $\mathbf{f}_i(\mathbf{v}_i)$, and $f(\mathbf{y})$ is a term in $\mathbf{f}_j(\mathbf{v}_j)$, for some function symbol f and $i, j \in \{1, \ldots, n\}$, $i \neq j$.

Statistics (S_1) and (S_2) are required by all three estimators that we have introduced, and can be measured directly by evaluating source queries on $\mathcal{D}$. Statistics (S_3) can be collected by first iterating over the function symbols in the mappings, and then calculating the cardinalities for joins over pairs of source queries whose corresponding mapping targets have a function symbol in common. It is easy to check that Statistics (S_1)–(S_3) suffice for our estimation, since all joins in a CQ are between source queries, and moreover, every translation calculated according to Definition 2.1 contains only joins between pairs of source queries considered by Statistics (S_3). The same intuitions apply for the possible projections applied to a basic CQ in an unfolding.

5.5 Cardinality Estimation of an Unfolding

We now show how to estimate the cardinality of an unfolding[4] by using the formulae (6), (7), and (8) introduced for cardinality estimation. The next theorem shows that such estimation can be calculated by summing-up the estimated cardinalities for each CQ in the unfolding of the input query, provided that *(i)* the unfolding is being calculated over the *wrap* of the set of mappings, and *(ii)* the query to unfold is a CQ.

We start our discussion with an auxiliary lemma about rules in an unfolding.

Lemma 5.5. *Consider an unfolding $(q(\mathbf{v}), \Pi)$ and two rules $r_1, r_2 \in \Pi$. If $head(r_1)$ and $head(r_2)$ are not unifiable, then for each database instance $\mathcal{D}$ for Σ, we have that*

$$(q(\mathbf{v}), \{r_1\})^{\mathcal{D}} \cap (q(\mathbf{v}), \{r_2\})^{\mathcal{D}} = \emptyset.$$

Proof. Let $A_1 = (q(\mathbf{v}), \{r_1\})^{\mathcal{D}}$, and $A_2 = (q(\mathbf{v}), \{r_2\})^{\mathcal{D}}$. W.l.o.g., let $head(r_1) = q(\mathbf{f}(\mathbf{x}))$, and $head(r_2) = q(\mathbf{g}(\mathbf{y}))$. Since $\mathbf{f}(\mathbf{x})$ and $\mathbf{g}(\mathbf{y})$ are not unifiable, it must be that $\mathbf{f} \neq \mathbf{g}$. Since no constants occur in either $head(r_1)$ or $head(r_2)$, it directly follows from the definition of answer of a Datalog query over an instance $\mathcal{D}$ that $A_1 \subseteq \{\mathbf{f}(\mathbf{a}) \mid \mathbf{a} \in adom(\mathcal{D})\}$, and $A_2 \subseteq \{\mathbf{g}(\mathbf{a}) \mid \mathbf{a} \in adom(\mathcal{D})\}$, where $adom(\mathcal{D})$ denotes the set of constants occurring in $\mathcal{D}$[5]. The thesis follows by observing that $\{\mathbf{f}(\mathbf{a}) \mid \mathbf{a} \in adom(\mathcal{D})\} \cap \{\mathbf{g}(\mathbf{a}) \mid \mathbf{a} \in adom(\mathcal{D})\} = \emptyset$. □

Theorem 5.6. *Consider a CQ $q(\mathbf{x}) \leftarrow L_1(\mathbf{v}_1), \ldots, L_n(\mathbf{v}_n)$ such that $\mathbf{x} = \bigcup_{i=1}^{n} \mathbf{v_i}$. Then,*

$$|unf(q(\mathbf{x}), \mathcal{M})^{\mathcal{D}}| = \sum_{q_u \in unf(q, wrap(\mathcal{M}))} |q_u(\mathbf{x})^{\mathcal{D}}|$$

Proof. Since by Lemma 4.9 $\mathcal{M} \equiv wrap(\mathcal{M})$, it suffices to prove that for each pair of different queries q_u, q'_u in $unf(q(\mathbf{x}), wrap(\mathcal{M}))$ it holds that $q_u^{\mathcal{D}} \cap q_u'^{\mathcal{D}} = \emptyset$. Consider two arbitrary queries q_u, q'_u in $unf(q(\mathbf{x}), wrap(\mathcal{M}))$. By Definition 2.1, there must exist two different lists of mappings $(m_1, \ldots, m_n)$ and $(m'_1, \ldots, m'_n)$ in $wrap(\mathcal{M})$ and two substitutions σ, σ' such that:

(i) $head(q_u) = q_u(\sigma(\mathbf{x}))$, $head(q'_u) = q'_u(\sigma'(\mathbf{x}))$;

(ii) $target(m_i) = L_i(\sigma(\mathbf{v}_i))$, $target(m'_i) = L_i(\sigma'(\mathbf{v}_i))$, for each $i \in \{1, \ldots, n\}$.

W.l.o.g., let $m_i \neq m'_i$, for some $i \in \{1, \ldots, n\}$. Moreover, assume that $\sigma(\mathbf{v}_i) = \mathbf{f}_i(\mathbf{x}_i)$ and $\sigma'(\mathbf{v}_i) = \mathbf{f}'_i(\mathbf{x}'_i)$, for some tuples of terms $\mathbf{f}_i(\mathbf{x}_i)$ and $\mathbf{f}'_i(\mathbf{x}'_i)$. By *(ii)* and Lemma 4.10, it must be $\mathbf{f}_i \neq \mathbf{f}'_i$. Hence, $\mathbf{f}_i(\mathbf{x}_i)$ and $\mathbf{f}'_i(\mathbf{x}'_i)$ do not unify. Therefore, also $\sigma(\mathbf{v}_i)$ and $\sigma'(\mathbf{v}_i)$ do not unify. Since, by assumption, $\mathbf{x} \supseteq \mathbf{v_i}$, we conclude that $\sigma(\mathbf{x})$ does not unify with $\sigma'(\mathbf{x})$. Then, by Lemma 5.5, it follows that

$$q_u(\sigma(\mathbf{x}))^{\mathcal{D}} \cap q'_u(\sigma'(\mathbf{x}))^{\mathcal{D}} = \emptyset$$

By *(i)*, it holds that $q_u^{\mathcal{D}} \cap q_u'^{\mathcal{D}} = \emptyset$. Since the choice of q_u, q'_u was arbitrary, we conclude the thesis. □

[4] Regardless of whether the unfolding is of a UCQ or a JUCQ, as they *must* be equivalent.

[5] Called *active domain*, see [1]

The previous theorem states that the cardinality of the unfolding of a query over the wrap of a set of mappings corresponds to the sum of the cardinalities of each CQ in the unfolding, under the assumption that all the attributes are kept in the answer. Intuitively, the proof relies on the fact that, when the wrap of a set of mappings is used, each CQ in the unfolding returns answer variables built using a specific combination of function names. Hence, to calculate the cardinality of a CQ q, it suffices to collect statistics as described in the previous paragraph, but over $wrap(\mathcal{M})$ rather than $\mathcal{M}$, and sum up the estimations for each CQ in $unf(q, wrap(\mathcal{M}))$.

The method above might overestimate the actual cardinality if the input CQ contains non-answer variables. In Section 5.7 we show how to address this limitation by storing, for each property in the mappings, the probability of having duplicate answers if the projection operation is applied to one of the (two) arguments of that property.

Also, the method above assumes a CQ as input to the unfolding, whereas a perfect rewriting is in general a UCQ. This is usually not a critical aspect, especially in practical applications of OBDA. By using a saturated set of mappings (called T-mapping [20]) $\mathcal{M}_\mathcal{T}$ in place of $\mathcal{M}$, in fact, the perfect rewriting of an input CQ q *almost always* [15] coincides with q itself[6]. Hence, in most cases we can directly use in Theorem 5.6 the input query q, if we use $wrap(\mathcal{M}_\mathcal{T})$ instead of $wrap(\mathcal{M})$. In the following subsection we provide a fully detailed example of how this is done, as well as more details on T-mappings and related techniques.

5.6 A Note on Rewriting

Modern OBDA systems such as *Ontop* or Ultrawrap [23] use advanced rewriting techniques for the rewriting phase. In particular, *Ontop* uses a combination of the *T-mapping* and *tree-witness rewriting* [6] techniques. We now give basic notions about these two techniques that will be useful in the rest of this paper.

Definition 5.7. *The* saturation $\mathcal{A}_\mathcal{T}$ of an ABox $\mathcal{A}$ with respect to a TBox $\mathcal{T}$ *is the ABox*

$$\{L(\mathbf{a}) \mid (\mathcal{T}, \mathcal{A}) \models L(\mathbf{a}), \mathbf{a} \textit{ is a tuple of individuals}\}$$

In OBDA, saturated virtual ABoxes can obtained by compiling the ontology in the mappings so as to obtain a set of mappings called *T-mapping* [21]. Intuitively, a T-mapping exposes a saturated ABox.

Definition 5.8. *Given an OBDA specification $\mathcal{S} = (\mathcal{T}, \mathcal{M}, \Sigma)$, the mapping set of mappings $\mathcal{M}_\mathcal{T}$ is a T-mapping for $\mathcal{S}$ if, for every OBDA instance $\mathcal{D}$ of $\mathcal{S}$, $\mathcal{A}_{(\mathcal{M},\mathcal{D})_\mathcal{T}} = \mathcal{A}_{(\mathcal{M}_\mathcal{T},\mathcal{D})}$.*

The saturation of an ABox is also known as a *forward chaining* technique, as opposed to *backward chaining* techniques such as rewriting algorithms. It is easy to see that, for each saturated ABox $\mathcal{A}_\mathcal{T}$ and *single-atom query* $q(\mathbf{x}) \leftarrow X(\mathbf{x})$, it holds that $q(\mathbf{x})^{\mathcal{A}_\mathcal{T}} = cert(q(\mathbf{x}), (\mathcal{T}, \mathcal{A}))$. However, such equality does not hold in case q is a general CQ.

Authors from [12] gave a characterization of queries for which query answering with respect to an ontology corresponds to query evaluation over saturated ABoxes. The *tree-witness rewriting* technique [12] exploits this characterization so as to produce perfect rewritings that are minimal with respect to a saturated ABox. In short, queries for which query answering with respect to an ontology *does not* correspond to query evaluation over saturated ABoxes are the ones whose answers are influenced by existentials in the right-hand-side of *DL-Lite*$_\mathcal{R}$ TBox axioms. Such queries are called queries with *tree-witnesses*.

Example 5.9. Consider the data instance $\mathcal{D}$ from Figure 2. Consider the query

$$q(x, y, z) \leftarrow C(x), R_1(x, y), R_2(x, z)$$

and the set of mappings

$$\mathcal{M} = \left\{ \begin{array}{llll} A(l(a)) & \rightsquigarrow T_1(a) & P_2(l(g), m(h)) & \rightsquigarrow T_5(g, h) \\ C(l(b)) & \rightsquigarrow T_2(b) & P_2(l(g), n(h)) & \rightsquigarrow T_5(g, h) \\ P_1(l(c), m(d)) & \rightsquigarrow T_3(c, d) & R_2(l(i), m(j)) & \rightsquigarrow T_6(i, j) \\ R_1(l(e), m(f)) & \rightsquigarrow T_4(e, f) & & \end{array} \right\}$$

Let $\emptyset = \{A \sqsubseteq C, P_1 \sqsubseteq R_1, P_2 \sqsubseteq R_2\}$. Since all the variables in q are answer variables, to compute $cert(q, (\mathcal{T}, \mathcal{A}_{(\mathcal{M},\mathcal{D})})$ it suffices to compute the evaluation $q^{\mathcal{A}_{(\mathcal{M}_\mathcal{T},\mathcal{D})}}$, where $\mathcal{M}_\mathcal{T}$ is a T-mapping of $\mathcal{M}$ with respect to $\mathcal{T}$. Such T-mapping is:

$$\mathcal{M}_\mathcal{T} = \mathcal{M} \cup \left\{ \begin{array}{llll} C(l(a)) & \rightsquigarrow T_1(a) & R_2(l(c), m(h)) & \rightsquigarrow T_5(g, h) \\ R_1(l(c), m(d)) & \rightsquigarrow T_3(c, d) & R_2(l(g), n(h)) & \rightsquigarrow T_5(g, h) \end{array} \right\}$$

[6] *Always*, if the CQ is interpreted as a SPARQL query and evaluated according to the OWL 2 QL entailment regime, or if the CQ does not contain existentially quantified variables.

$\mathbf{T_1}$

a	...
1	...
2	...
3	...
4	...
5	...

$\mathbf{T_2}$

b	...
1	...
2	...
3	...
4	...
5	...

$\mathbf{T_3}$

c	d	...
1	1	...
2	3	...
3	5	...
4	7	...
5	9	...
6	11	...
7	13	...
8	15	...
9	17	...
10	19	...

$\mathbf{T_4}$

e	f	...
1	1	...
2	3	...
3	5	...
4	7	...
5	9	...
6	11	...
7	13	...
8	15	...
9	17	...
10	19	...
11	21	...

$\mathbf{T_5}$

g	h	...
1	2	...
2	4	...
3	6	...
4	8	...
5	10	...
6	12	...
7	14	...
8	16	...
9	18	...
10	20	...

$\mathbf{T_6}$

i	j	...
1	2	...
2	4	...
3	6	...
4	8	...
5	10	...
6	12	...
7	14	...
8	16	...
9	18	...
10	20	...
11	22	...

Figure 2: Data Instance $\mathcal{D}$.

The wrap $wrap(\mathcal{M}_\mathcal{T})$ of $\mathcal{M}_\mathcal{T}$ is the set

$$\left\{\begin{array}{llll} A(l(a)) & \leftsquigarrow T_1(a) & P_2(l(g), m(h)) & \leftsquigarrow T_5(g,h) \\ C(l(v)) & \leftsquigarrow V_1(v) & P_2(l(g), n(h)) & \leftsquigarrow T_5(g,h) \\ P_1(l(c), m(d)) & \leftsquigarrow T_3(c,d) & R_2(l(z_1), m(z_2)) & \leftsquigarrow V_3(z_1, z_2) \\ R_1(l(w_1), m(w_2)) & \leftsquigarrow V_2(w_1, w_2) & R_2(l(g), n(h)) & \leftsquigarrow T_5(g,h) \end{array}\right\}$$

where $V_1, \ldots, V_3$ are view names for Datalog queries of programs respectively

$$\begin{array}{lcl} \Pi_1 & = & \{V_1(a) \leftarrow T_1(a), V_1(b) \leftarrow T_2(b)\} \\ \Pi_2 & = & \{V_2(c,d) \leftarrow T_3(c,d), V_2(e,f) \leftarrow T_4(e,f)\} \\ \Pi_3 & = & \{V_3(g,h) \leftarrow T_5(g,h), V_3(i,j) \leftarrow T_6(i,j)\}. \end{array}$$

To estimate the cardinality of $q^{\mathcal{A}_{(\mathcal{M}_\mathcal{T}, \mathcal{D})}}$ by using our cardinality estimator, we need the following statistics:

- $|\pi_v(V_1)| = 5$
- $|\pi_{w_1,w_2}(V_2)| = 11$
- $|\pi_{z_1,z_2}(V_3)| = 11$
- $|\pi_{g,h}(T_5)| = 10$
- $|\pi_{w_1}(V_2)| = |\pi_{w_2}(V_2)| = |\pi_{z_1}(V_3)| = |\pi_{z_2}(V_3)| = 11$

- $|\pi_i(T_6)| = |\pi_j(T_6)| = 10$
- $|\pi_v(V_1) \Cap \pi_{w_1}(V_2)| = |\pi_v(V_1) \Cap \pi_{z_1}(V_3)| = 5$
- $|\pi_v(V_1) \Cap \pi_g(T_5)| = 5$

By applying Definition 2.1, the unfolding $unf(q, wrap(\mathcal{M}_\mathcal{T}))$ is the NRLP query $(q(x, y, z), \Pi_{unf})$, where:

$$\Pi_{unf} = \left\{ \begin{array}{lcl} q(l(v), m(w_2), m(z_2)) & \leftarrow & V_1(v), V_2(v, w_2), V_3(v, z_2) \\ q(l(v), m(w_2), n(h)) & \leftarrow & V_1(v), V_2(v, w_2), T_5(v, h) \end{array} \right\}$$

In relational algebra:

$$E_1 = \pi_{x/l(v), y/m(w_2), z/m(z_2)}(V_1 \bowtie_{V_1.v=V_2.w_2} V_2 \bowtie_{V_1.v=V_3.z_2} V_3)$$
$$\cup$$
$$E_2 = \pi_{x/l(v), y/m(w_2), z/n(h)}(V_1 \bowtie_{V_1.v=V_2.w_2} V_2 \bowtie_{V_1.v=T_5.a} T_5)$$

According to Theorem 5.6, the cardinality of q_{unf} in $\mathcal{D}$ can be calculated as $|E_1| + |E_2|$. Such cardinalities can be computed in the same fashion as in Example 5.3.

5.7 Extending Cardinality Estimation to Input Queries with Projection

In the presence of projection in the input query, our estimation for the cardinality of an unfolding is an upper-bound of the actual cardinality, if the cardinalities for each single CQ are estimated correctly. In this section we discuss a lower-bound of the actual cardinality of an unfolding.

We first recall how to compute the cardinality of a union of sets. Given sets A, B, the following equation holds:

$$|A \cup B| = |A| + |B| - |A \cap B|. \tag{16}$$

In general, given n sets $A_1, \ldots, A_n$, it holds that

$$\left|\bigcup_i A_i\right| \;=\; \sum_{i=1}^{n} \left(\sum_{\substack{J \subseteq \{1,\ldots,n\} \\ |J|=i}} ((-1)^{i-1} \cdot \left|\bigcap_{j \in J} A_j\right|) \right). \tag{17}$$

Recall that our estimator stores the cardinalities of the intersections of *pairs of joinable columns*. An idea could be to estimate the cardinality of a union between n sets by exploiting such cardinalities. Observe that, according to the equations above, the information about the cardinality of the intersection of pairs of sets is sufficient to calculate the cardinality of the union of two sets. However, it is not sufficient to calculate the cardinality of the union of more than two sets. Still, one can estimate a lower bound for such cardinality, by assuming that for three arbitrary sets A, B, C, we have that

$$|A \cap B \cap C| \;=\; \min\{|A \cap B|, |A \cap C|, |B \cap C|\}. \tag{a4}$$

Intuitively, we are assuming that elements are shared *as much as possible* between the three sets A, B, C.

Following this idea, in the next example we show how to calculate the cardinality of a union of n sets by using a set of linear equations, and by relying only on the information about the cardinalities of the intersections of pairs of sets.

Example 5.10. Consider the sets $A = \{1, 2, 3, 4\}$, $B = \{1, 2, 5, 6\}$, $C = \{1, 2, 3, 4\}$, and $D = \{5, 6, 7, 8\}$. We consider the following cardinalities to be available:

- $|A| = |B| = |C| = |D| = 4$
- $|A \cap B| = 2$, $|A \cap C| = 4$, $|A \cap D| = 0$, $|B \cap C| = 2$, $|B \cap D| = 2$, $|C \cap D| = 0$.

For each expression of the form $|S_1 \cap \cdots \cap S_k|$, we introduce a variable $x_{S_1 \cdots S_k}$.

In order to calculate $|A \cup B \cup C \cup D|$, by applying Equation (17) we first need to calculate $|A \cap B \cap C|$, $|A \cap B \cap D|$, $|B \cap C \cap D|$, $|A \cap C \cap D|$, and $|A \cap B \cap C \cap D|$. Boundaries for such values can be found by using the following system of linear inequalities:

$$\begin{array}{l} x_{ABC} \leq \min\{x_{AB}, x_{AC}, x_{BC}\} \\ x_{ABD} \leq \min\{x_{AB}, x_{AD}, x_{BD}\} \\ x_{BCD} \leq \min\{x_{BC}, x_{CD}, x_{BD}\} \\ x_{ACD} \leq \min\{x_{AC}, x_{CD}, x_{AD}\} \\ x_{ABCD} \leq \min\{x_{ABC}, x_{ABD}, x_{BCD}, x_{ACD}\} \\ x_{AB} = 2, x_{AC} = 4, x_{AD} = 0, x_{BC} = 2, x_{BD} = 2, x_{CD} = 0 \end{array}$$

By Assumption (a4), we obtain:

$$
\begin{aligned}
&x_{ABC} = \min\{x_{AB}, x_{AC}, x_{BC}\} = \min\{2, 4, 2\} = 2\\
&x_{ABD} = \min\{x_{AB}, x_{AD}, x_{BD}\} = \min\{2, 0, 2\} = 0\\
&x_{BCD} = \min\{x_{BC}, x_{CD}, x_{BD}\} = \min\{2, 0, 2\} = 0\\
&x_{ACD} = \min\{x_{AC}, x_{CD}, x_{AD}\} = \min\{4, 0, 0\} = 0\\
&x_{ABCD} = \min\{x_{ABC}, x_{ABD}, x_{BCD}, x_{ACD}\} = \min\{2, 0, 0, 0\} = 0\\
&x_{AB} = 2, x_{AC} = 4, x_{AD} = 0, x_{BC} = 2, x_{BD} = 2, x_{CD} = 0
\end{aligned}
$$

By applying the results of the system in Equation (17), we obtain:

$$4 \cdot 4 - (2 + 4 + 0 + 2 + 2 + 0) + (2 + 0 + 0 + 0) - 0 = 16 - 10 + 2 = 8$$

Observe that, in this example, the obtained lower bound 8 corresponds to the actual cardinality of $A \cup B \cup C \cup D$. ■

Observe that the previous approach for calculating the cardinality of a union of sets comes at the cost of calculating an expression of length exponential in the size of the union. Thus, this approach is feasible only for unions of a small number of sets.

We now show how the approach above, or any other approach able to estimate the cardinality for the union of sets, can be used to calculate the cardinality of an unfolding in the presence of projection. To do so, we first need to introduce the auxiliary notion of *answer template matrix*.

Definition 5.11. *Let $q(\mathbf{x}) \leftarrow L_1(\mathbf{v}_1), \ldots, L_n(\mathbf{v}_n)$ be a CQ, and consider the unfolding $unf(q, \mathcal{M}) = (q_{unf}(\mathbf{x}), \Pi)$ of q with respect to $\mathcal{M}$. W.l.o.g., we assume Π to be of the form*

$$\Pi = \left\{ q_{unf}(\mathbf{f}_j(\mathbf{y}_j)) \leftarrow V_1^j \wedge \cdots \wedge V_n^j \;\middle|\; 1 \leq j \leq m \right\}.$$

Then, the answer template matrix (ATM) *of q_{unf} is the matrix*

$$atm(q_{unf}) = \begin{bmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_m \end{bmatrix}$$

of tuples of function templates $\mathbf{f}_1, \ldots, \mathbf{f}_m$.

It is possible to use the notion of ATM to improve our estimation for the cardinality of an unfolding in the presence of the projection operator. The estimation proceeds differently based on whether *(i)* the ATM does not contain duplicate rows, or *(ii)* the ATM contains duplicate rows.

We start our discussion with case *(i)*.

Estimation without Duplicate Rows in the ATM. By arguing as in the proof of Theorem 5.6, it is immediate to see that the ATM of an unfolding of a CQ q with respect to $wrap(\mathcal{M})$ *does not* contain repeated rows if q does not contain existentially quantified variables. By exploiting the notion of ATM, the next result extends Theorem 5.6 to a more general case.

Theorem 5.12. *Consider a CQ $q(\mathbf{x}) \leftarrow L_1(\mathbf{v}_1), \ldots, L_n(\mathbf{v}_n)$. Then, if $atm(unf(q(\mathbf{x}), wrap(\mathcal{M})))$ does not contain repeated rows, the following holds:*

$$|unf(q(\mathbf{x}), \mathcal{M})^{\mathcal{D}}| = \sum_{q_u \in unf(q, wrap(\mathcal{M}))} |q_u(\mathbf{x})^{\mathcal{D}}|$$

Proof. Since by Lemma 4.9 $\mathcal{M} \equiv wrap(\mathcal{M})$, it suffices to prove that for each pair of different queries q_u, q'_u in $unf(q(\mathbf{x}), wrap(\mathcal{M}))$ it holds that $q_u^{\mathcal{D}} \cap q_u'^{\mathcal{D}} = \emptyset$. Consider two arbitrary queries q_u, q'_u in $unf(q(\mathbf{x}), wrap(\mathcal{M}))$. By Definition 2.1, there must exist two different lists of mappings $(m_1, \ldots, m_n)$ and $(m'_1, \ldots, m'_n)$ in $wrap(\mathcal{M})$ and two substitutions σ, σ' such that

$$head(q_u) = q_u(\sigma(\mathbf{x})) \qquad \text{and} \qquad head(q'_u) = q'_u(\sigma'(\mathbf{x})). \tag{18}$$

Since the ATM of the unfolding does not contain repeated rows, then necessarily $\sigma(\mathbf{x})$ and $\sigma'(\mathbf{x})$ are not unifiable. Then, by Lemma 5.5, it follows that

$$q_u(\sigma(\mathbf{x}))^{\mathcal{D}} \cap q'_u(\sigma'(\mathbf{x}))^{\mathcal{D}} = \emptyset$$

By Equations (18), it holds that $q_u^{\mathcal{D}} \cap q_u'^{\mathcal{D}} = \emptyset$. Since the choice of q_u, q'_u was arbitrary, we conclude the thesis. □

Theorem 5.12 gives us a less restrictive condition for calculating the cardinality of an unfolding, as it essentially says that projecting out variables does not change the number of results as long as duplicate rows do not appear in the ATM of the unfolding.

Estimation with Duplicate Rows in the ATM. We now study the problem of estimating the number of results of an unfolding for which the ATM contains duplicate rows. First, observe that any unfolding can be partitioned into several different UCQs where either the ATM *does not* contain duplicate rows, or all rows are equal. Summing up the cardinalities of such partitions would give the cardinality of the original unfolding. Hence, we can calculate the cardinality of a general unfolding by handling such two cases. In the previous paragraph we have already seen how to deal with the case in which there are no duplicate rows in the ATM. In this paragraph we study the case in which all rows of the ATM are equal. For this we exploit another useful property of the unfolding with respect to the wrap of a set of mappings.

Theorem 5.13. *Let $q(\mathbf{x}) \leftarrow L_1(\mathbf{v}_1), \ldots, L_n(\mathbf{v}_n)$ be a CQ for which the unfolding $unf(q, wrap(\mathcal{M}))$ of q with respect to $wrap(\mathcal{M})$ is such that all rows in the ATM are equal to some tuple $\mathbf{f}$ of templates. Let $L_i(\mathbf{v}_i)$ be an atom in q such that $\mathbf{x} \supseteq \mathbf{v}_i$. Then, for each pair of rules*

$$\begin{array}{rrcl} r_1 = & q_u(\mathbf{f}(\mathbf{y})) & \leftarrow & V_1(\mathbf{y}_1), \ldots, V_i(\mathbf{y}_i), \ldots, V_n(\mathbf{y}_n), \\ r_2 = & q'_u(\mathbf{f}(\mathbf{y}')) & \leftarrow & V'_1(\mathbf{y}'_1), \ldots, V'_i(\mathbf{y}'_i), \ldots, V'_n(\mathbf{y}'_n) \end{array}$$

in the program of $unf(q, wrap(\mathcal{M}))$, it holds that $V_i = V'_i$.

Proof. By Definition 2.1, there must exist two different lists of mappings $(m_1, \ldots, m_n)$ and $(m'_1, \ldots, m'_n)$ in $wrap(\mathcal{M})$ and two substitutions σ, σ' such that:

(i) $q_u(\mathbf{f}(\mathbf{y})) = q_u(\sigma(\mathbf{x}))$, $q'_u(\mathbf{f}(\mathbf{y}')) = q'_u(\sigma'(\mathbf{x}))$;

(ii) $target(m_j) = L_j(\sigma(\mathbf{v}_j))$, $target(m'_j) = L_j(\sigma'(\mathbf{v}_j))$, for each $j \in \{1, \ldots, n\}$;

(iii) $\sigma(source(m_j)) = V_j(\mathbf{y}_j)$, $\sigma'(source(m'_j)) = V'_j(\mathbf{y}'_j)$, for each $j \in \{1, \ldots, n\}$.

By *(iii)*, in order to show that $V_i = V'_i$, it suffices to prove that $m_i = m'_i$. Assume by contradiction that $m_i \neq m'_i$. Let $\sigma(\mathbf{v}_i) = \mathbf{f}_i(\mathbf{x}_i)$ and $\sigma'(\mathbf{v}_i) = \mathbf{f}'_i(\mathbf{x}'_i)$, where $\mathbf{f}_i(\mathbf{x}_i)$ and $\mathbf{f}'_i(\mathbf{x}'_i)$ are tuples of terms. By *(ii)* and Lemma 4.10, it must be $\mathbf{f}_i \neq \mathbf{f}'_i$. Hence, $\mathbf{f}_i(\mathbf{x}_i)$ and $\mathbf{f}'_i(\mathbf{x}'_i)$ do not unify. Therefore, also $\sigma(\mathbf{v}_i)$ and $\sigma'(\mathbf{v}_i)$ do not unify. Since, by assumption, $\mathbf{x} \supseteq \mathbf{v}_i$, we conclude that $\sigma(\mathbf{x})$ does not unify with $\sigma'(\mathbf{x})$. However, this contradicts with *(i)*. Hence, $m_i = m'_i$. □

In the next example we show how to exploit Therem 5.13 for the estimation of the cardinality of an unfolding.

Example 5.14. Consider a database instance $\mathcal{D}$ and the CQ $q(x, y) \leftarrow P_1(x, y), P_2(y, z), P_3(z, w)$, for which all rows in the ATM of $unf(q, wrap(\mathcal{M})) = (q_{unf}(x, y), \Pi)$ are equal. Then, by Theorem 5.13 above, Π is of the following shape:

$$\begin{array}{rrcl} cq_1: & q_{unf}(f(x), g(y)) & \leftarrow & V_{P_1}(x, y), V^1_{P_2}(y, z_1), V^1_{P_3}(z_1, w_1) \\ & & & \vdots \\ cq_n: & q_{unf}(f(x), g(y)) & \leftarrow & V_{P_1}(x, y), V^n_{P_2}(y, z_n), V^n_{P_3}(z_n, w_n) \end{array}$$

That is, the relation V_{P_1} appears in each $cq_1, \ldots, cq_n$. Due to this fact, together with the fact that atoms $V^1_{P_3}(z_1, w_1), \ldots, V^n_{P_3}(z_n, w_n)$ do not contain answer variables, we have the interesting property that any duplicate *across different CQs* in the occurrences of y in $V^j_{P_2}$ (marked in blue), for $j \in \{1, \ldots, n\}$, would produce a duplicate result in $q^{\mathcal{D}}$. Under Assumption (a2) of uniformity, we get

$$|q(x, y)^{\mathcal{D}}| \;=\; \frac{|(\pi_y(cq^*_1) \cup \cdots \cup \pi_y(cq^*_n))^{\mathcal{D}}|}{\sum_{i=1}^{n} |(\pi_y(cq^*_i))^{\mathcal{D}}|} \cdot \sum_{j=1}^{n} |cq^{\mathcal{D}}_j|,$$

where cq^*_i represents the query cq_i in which we have replaced in the head $f(x)$ with x and $g(y)$ with y, and the fraction

$$\frac{|(\pi_y(cq^*_1) \cup \cdots \cup \pi_y(cq^*_n))^{\mathcal{D}}|}{\sum_{i=1}^{n} |(\pi_y(cq^*_i))^{\mathcal{D}}|}$$

denotes the ratio of distinct values across the different CQs over the answer variable y, calculated by assuming uniformity, i.e., that each value is repeated the same amount of times. ■

For estimating the cardinality of the unfolding from the previous example, we need to be able estimate the quantity $|(\pi_y(cq^*_1) \cup \cdots \cup \pi_y(cq^*_n))^{\mathcal{D}}|$. For this, we can resort to Equation (17) as in Example 5.10, by using the available statistics on intersections between pairs of attributes sets that are arguments for some function symbol. We also observe that, in the case where $V^j_{P_2} = V^k_{P_2}$ for some j and k, then $|(\pi_y(cq^*_j) \cup \pi_y(cq^*_k))^{\mathcal{D}}| = \max\{|(\pi_y(cq^*_j))^{\mathcal{D}}|, |(\pi_y(cq^*_k))^{\mathcal{D}}|\}$, and the computation of the cardinality of the union can be simplified.

Example 5.15. Recall the scenario from Example 5.14. Assume that $n = 6$, and that

$$V^1_{P_2} = V^2_{P_2}$$
$$V^3_{P_2} = V^4_{P_2}$$
$$V^5_{P_2} = V^6_{P_2}$$

Then, let

$$\begin{array}{rcl} |(\pi_y(cq_1^*))^{\mathcal{D}}| & = & \max\{|(\pi_y(cq_1^*))^{\mathcal{D}}|, |(\pi_y(cq_2^*))^{\mathcal{D}}|\} \; = \; |(\pi_y(cq_1^*) \cup \pi_y(cq_2^*))^{\mathcal{D}}| \\ |(\pi_y(cq_3^*))^{\mathcal{D}}| & = & \max\{|(\pi_y(cq_3^*))^{\mathcal{D}}|, |(\pi_y(cq_4^*))^{\mathcal{D}}|\} \; = \; |(\pi_y(cq_3^*) \cup \pi_y(cq_4^*))^{\mathcal{D}}| \\ |(\pi_y(cq_5^*))^{\mathcal{D}}| & = & \max\{|(\pi_y(cq_5^*))^{\mathcal{D}}|, |(\pi_y(cq_6^*))^{\mathcal{D}}|\} \; = \; |(\pi_y(cq_5^*) \cup \pi_y(cq_6^*))^{\mathcal{D}}|. \end{array}$$

Hence,

$$|(\pi_y(cq_1^*) \cup \cdots \cup \pi_y(cq_6^*))^{\mathcal{D}}| = |(\pi_y(cq_1^*) \cup \pi_y(cq_3^*) \cup \pi_y(cq_5^*))^{\mathcal{D}}|.$$

By using Equation (17), we get the formula:

$$\begin{array}{rl} |(\pi_y(cq_1^*) \cup \pi_y(cq_3^*) \cup \pi_y(cq_5^*))^{\mathcal{D}}| = & |(\pi_y(cq_1^*))^{\mathcal{D}}| + |(\pi_y(cq_3^*))^{\mathcal{D}}| + |(\pi_y(cq_5^*))^{\mathcal{D}}| \\ & - |(\pi_y(cq_1^*) \cap \pi_y(cq_3^*))^{\mathcal{D}}| - |(\pi_y(cq_1^*) \cap \pi_y(cq_5^*))^{\mathcal{D}}| - |(\pi_y(cq_3^*) \cap \pi_y(cq_5^*))^{\mathcal{D}}| \\ & + |(\pi_y(cq_1^*) \cap \pi_y(cq_3^*) \cap \pi_y(cq_5^*))^{\mathcal{D}}|. \end{array} \tag{19}$$

Assume that

$$\max\{|(\pi_y(cq_1^*) \cap \pi_y(cq_3^*))^{\mathcal{D}}|, |(\pi_y(cq_1^*) \cap \pi_y(cq_5^*))^{\mathcal{D}}|, |(\pi_y(cq_3^*) \cap \pi_y(cq_5^*))^{\mathcal{D}}|\} = |(\pi_y(cq_1^*) \cap \pi_y(cq_3^*))^{\mathcal{D}}|.$$

Then, by relying on Assumption (a4), Equation (19) can be rewritten as:

$$\begin{array}{rl} |(\pi_y(cq_1^*) \cup \pi_y(cq_3^*) \cup \pi_y(cq_5^*))^{\mathcal{D}}| = & |(\pi_y(cq_1^*))^{\mathcal{D}}| + |(\pi_y(cq_3^*))^{\mathcal{D}}| + |(\pi_y(cq_5^*))^{\mathcal{D}}| \\ & - 2 \cdot |(\pi_y(cq_1^*) \cap \pi_y(cq_3^*))^{\mathcal{D}}| - |(\pi_y(cq_1^*) \cap \pi_y(cq_5^*))^{\mathcal{D}}| - |(\pi_y(cq_3^*) \cap \pi_y(cq_5^*))^{\mathcal{D}}|. \end{array}$$

We use such formula, containing only binary intersections, to compute the ratio of distinct values

$$r \; = \; \frac{|(\pi_y(cq_1^*) \cup \pi_y(cq_3^*) \cup \pi_y(cq_5^*))^{\mathcal{D}}|}{|(\pi_y(cq_1^*))^{\mathcal{D}}| + |(\pi_y(cq_3^*))^{\mathcal{D}}| + |(\pi_y(cq_5^*))^{\mathcal{D}}|}.$$

Hence, we estimate $|q(x, y)^{\mathcal{D}}|$ to be

$$r \cdot \sum_{j=1}^{6} |cq_j^{\mathcal{D}}| \tag{20}$$

■

The previous example makes use of Equation (17), which is practical only if the union is on a small number of sets. Additionally, for the sake of clarity, we have used the actual cardinalities, rather than estimates. We observe, however, that all the cardinalities in Formula (20) can be estimated by using our estimators $ce_{\mathcal{D}}$, $fv_{\mathcal{D}}$, and $dist_{\mathcal{D}}$.

6 Unfolding Cost Model

We are now ready to estimate the actual costs of evaluating UJUCQ and UCQ unfoldings, by exploiting the cardinality estimations from the previous section. Our cost model is based on traditional textbook-formulae for query cost estimation [24], and it assumes source parts in the mappings to be CQs.

6.1 Cost for the Unfolding of a UCQ.

Recall from Section 4 that the unfolding of a UCQ produces a UCQ translation $q^{ucq} = \bigvee_i q_i^{cq}$. We estimate the cost of evaluating q^{ucq} as

$$c_{eval}(q^{ucq}) \; = \; \sum_i c_{eval}(q_i^{cq}) + c_u(q^{ucq})$$

where

- $c_{eval}(q_i^{cq})$ is the cost of evaluating each q_i^{cq} in q^{ucq};
- $c_u(q^{ucq})$ is the cost of removing duplicate results.

In the remainder of this paragraph we explain how we calculate each of these addends.

Value of $c_{eval}(q^{cq})$. Let $q^{cq} = V_1 \bowtie \cdots \bowtie V_n$, where $V_i = T_{i1} \bowtie \cdots \bowtie T_{im_i}$, where $m_i \in \mathbb{N}$ and $1 \leq i \leq n$. Then, we define $c_{eval}(q^{cq})$ as

$$c_{eval}(q^{cq}) = \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m_i} c_{scan}(T_{ij}) + c_{hjoin}(q^{cq})$$

where

- $c_{scan}(T_{ij}) = |T_{ij}^{\mathcal{D}}| \times c_t$
  - where c_t is the fixed cost of retrieving one tuple from the database. Such constant, as well as other constants in this section, can be found through *calibration techniques* [11].
- $c_{hjoin}(q_i^{cq}) = (\sum_{i=1}^{n} m_i) \cdot ce_{\mathcal{D}}(q_i^{cq}) \cdot c_j$
  - where c_j is the fixed cost of joining one tuple.

Summing up, $c_{eval}(q_i^{cq})$ is the cost of scanning each table in the conjunctive query and performing a *hash join*. We assume the statistics $|T_{ij}^{\mathcal{D}}|$, for $1 \leq i \leq n$, $1 \leq j \leq m_i$, to be available after having analyzed the database instance according to the database schema.

Value of $c_u(q^{ucq})$. We assume the removal of duplicates at this level to be carried out by a sort-based strategy. Under this assumption, the cost of removal is

$$c_u(q^{ucq}) = ce_{\mathcal{D}}(q^{ucq}) \cdot log(ce_{\mathcal{D}}(q^{ucq})) \cdot c_u$$

where $ce_{\mathcal{D}}(q^{ucq})$ is the cardinality estimation of the unfolding, calculated as described in Section 5, and c_u is a constant denoting the cost of eliminating a duplicate tuple.

Cost for the Unfolding of a JUCQ. Recall from Section 4 that the optimized unfolding of a JUCQ produces a UJUCQ. We estimate the cost of a single JUCQ $q^{jucq} = \bigwedge_i q_i^{ucq}$ in the unfolding as

$$c(q^{jucq}) \;\; = \;\; \textstyle\sum_i c_{eval}(q_i^{ucq}) + \sum_{i \neq k} c_{mat}(q_i^{ucq}) + c_{mjoin}(q^{jucq})$$

where

- $c_{eval}(q_i^{ucq})$ is the cost of evaluating each UCQ component q_i^{ucq};
- $\sum_{i \neq k} c_{mat}(q_i^{ucq})$ is the cost of materializing the intermediate results from q_i^{ucq}, where the k-th UCQ is assumed to be *pipelined* [24] and not materialized;
- $c_{mjoin}(q^{jucq})$ is the cost of a merge join over the materialized intermediate results.

The cost for a UJUCQ $q^{ujucq} = \bigvee_i q_i^{jucq}$, if all the attributes are kept in the answer, is simply the sum $\sum_i c(q_i^{jucq})$, since the results of all JUCQs are disjoint (c.f., Section 4). Otherwise, we need to consider the cost of eliminating duplicate results.

Value of $c_{mat}(q_i^{ucq})$. Let $q_i^{ucq} = q_1^{cq} \bowtie \cdots \bowtie q_m^{cq}$. Then

$$c_{mat}(q_i)^{ucq} = ce_{\mathcal{D}}(q_i^{ucq}) \cdot c_m$$

where c_m is the fixed cost of materializing a tuple.

Value of $c_{mjoin}(q^{jucq})$. Let $q^{jucq} = q_1^{ucq} \bowtie \cdots \bowtie q_m^{ucq}$. Then

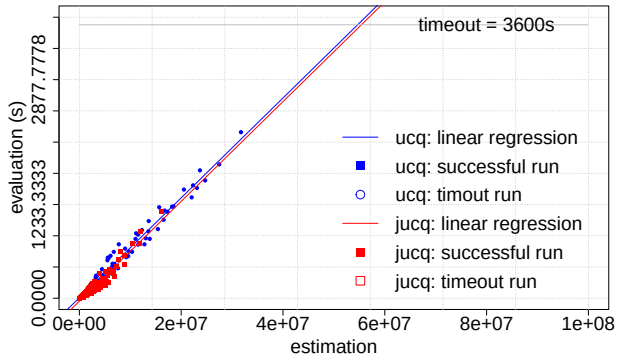
$$c_{mjoin}(q^{jucq}) = m \cdot ce_{\mathcal{D}}(q^{jucq}) \cdot c_j$$

where $ce_{\mathcal{D}}(q^{jucq})$ is the cardinality estimation of the unfolding, calculated as described in Section 5. Observe that we do not consider the cost of sorting each q_i^{ucq}, as this cost is already included in $c_{eval}(q_i^{ucq})$.

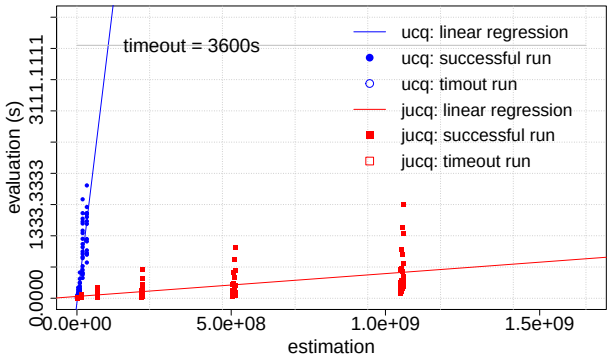
7 Experimental Results

In this section, we provide an empirical evaluation that compares unfoldings of UCQs and (optimized) unfoldings of JUCQs, as well as the estimated costs and the actual time needed to evaluate the unfoldings. We ran the experiments on an HP Proliant server with 2 Intel Xeon X5690 Processors (each with 12 logical cores at 3.47GHz), 106GB of RAM and five 1TB 15K RPM HDs. As RDBMS we have used PostgreSQL 9.6. The material to replicate our experiments is available online [7].

[7] https://github.com/ontop/ontop-examples/tree/master/iswc-2017-cost



(a) Our cost model vs. evaluation



(b) PostgreSQL cost model vs. evaluation

	queries		linear regression of our cost		linear regression of PostgreSQL cost	
	succ.	time out	b_0	b_1	b_0	b_1
UCQ	68	16	1.40e+01	6.17e-05	6.16e+01	3.61e+05
JUCQ	83	1	-3.76e+01	6.33e-05	2.56e+01	3.25e-07

(c) Results of linear regressions ($evaluation = b_0 + b_1 \times estimation$)

Figure 3: Cost estimations vs evaluation running times

7.1 Wisconsin Experiment

We devised an OBDA benchmark based on the *Wisconsin Benchmark* [9]. In the following subsections we discuss each element of the benchmark, and their rationale.

Mappings. The mappings set consists of 1364 mapping assertions. Each mapping defines a property of the form `:J20Oxx`M*y*`R`*z*`Prop`*i*, where

- `J20`, read "join selectivity 20%", denotes that each mapping for that property is on a query retrieving the 20% of the total tuples;
- Oxx, read "offset xx", where $xx \in \{0, 5, 10, 15\}$, denotes the offset for the filter in the mappings based on column onePercent;
- My, $y \in \{1, \ldots, 6\}$ denotes the number of mapping assertions defining the property;
- Rz, $z \leq y$ denotes the number of redundant mapping assertions, that is, the number of mapping assertions whose source query does not produces new individuals for the property in the virtual RDF ABox;
- Prop$i$, $i \in \{1, \ldots, 3\}$. The tested BGP is made of three properties.

For instance, we the following listing provides the mapping definition for the property `:S20O0M2R0Prop1`.

```
target      :number/{unique2} :S20O0M2R0Prop1
               :name/{evenOnePercent}/{stringu1}/{stringu2} .
source      SELECT "unique2", evenOnePercent, stringu1, stringu2
            FROM t1_1m
            WHERE "onepercent" >= 0 AND "onepercent" < 20
```

SPARQL Queries. Our test is on 84 queries, instantiations of the following template:

```
SELECT DISTINCT * WHERE {
   ?x :MmRrProp1 ?y1; :JjMmRrProp2 ?y2; :JjMmRrProp3 ?y3
}
```

where $j \in \{5, 10, 15, 20\}$ denotes the selectivity of the join between the first property and each of the remaining two, expressed as a percentage of the number of retrieved rows for each mapping defining the property (each mapping retrieves 200k tuples); $m \in \{1, \ldots, 6\}$ denotes the number of mappings defining the property (all such mappings have the same signature), and $r \in \{0, \ldots, m-1\}$ denotes the number of *redundant* mappings, that is, the number of mappings assertions retrieving the same results of another mapping defininining the property, minus one.

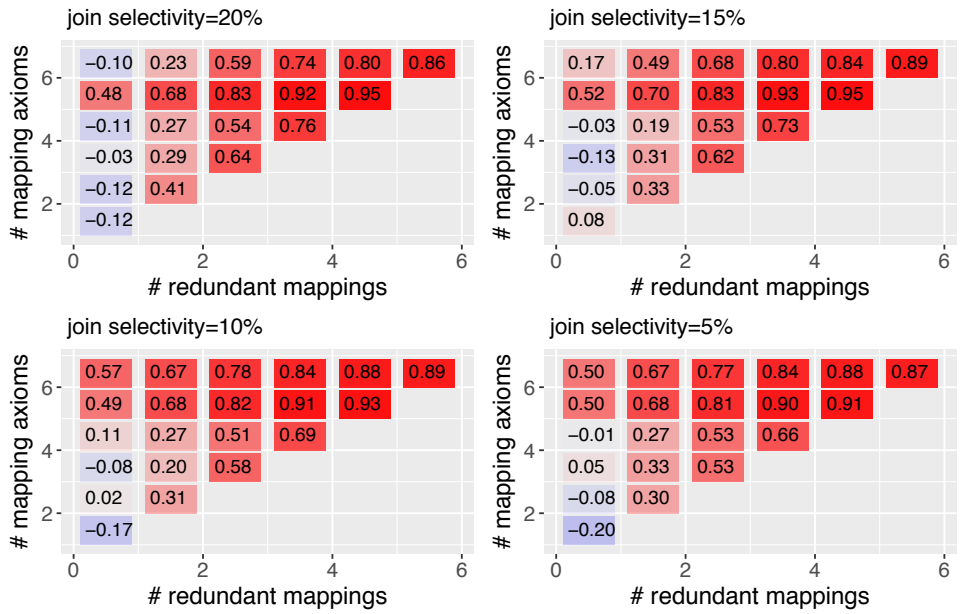


Figure 4: Performance gain of JUCQ compared with UCQ

For each query, we have tested a correspondent cover query of two fragments f_1, f_2, where each fragment is an instantiation of the following templates:

```
f1: SELECT DISTINCT ?x ?y1 ?y2 WHERE { ?x :MmRrProp1 ?y1; :JjMmRrProp2 ?y2. }
f2: SELECT DISTINCT ?x ?y3 WHERE { ?x :MmRrProp1 ?y2; ?x :JjMmRrProp3 ?y3. }
```

Ontology. We have carried out our tests over an empty TBox. Observe that this is not a limitation, as the case with a TBox can be reduced to the case without a TBox by making use of a T-mapping as in Example 5.9.

Data. We have created several copies of the wisconsin table, and populated each copy with ten million tuples.

Evaluation. In Figure 3, we present the cost estimation and the actual running time for each query. We have the following observations:

- In this experiment, for the considered cover, JUCQs are generally faster than UCQs. In fact, out of the 84 SPARQL queries, only one JUCQ was timed out, while 16 UCQs were timed out. The mean running time of successful UCQs and JUCQs are respectively 160 seconds and 350 seconds.
- In Figure 3a, where the fitted lines are obtained by applying linear regression over successful UCQ and JUCQ evaluations, we observe a strong linear correlation between our estimated costs and real running times. Moreover, the coefficients (b_1 and b_0) for UCQs and JUCQs are rather close. This empirically shows that our cost model can estimate the real running time well.
- Figure 3b shows that the PostgreSQL cost model assigns the same estimation to many queries having different running times. Moreover, the linear regressions for UCQs and JUCQs are rather different, which suggests that PostgreSQL is not able to recognize when two translations are semantically equivalent. Hence, PostgreSQL is not able to estimate the cost of these queries properly.

In Figure 4, we visualize the performance gain of JUCQs compared with UCQs. The four subgraphs correspond to four different settings join selectivities. Each subgraph is a matrix in which each cell shows the value of the performance gain $g = 1 - \text{jucq\_time}/\text{ucq\_time}$. When $g > 0$, we apply the red color; otherwise blue. These graphs clearly show that when there is a large number of mappings and there is high redundancy, we have better performance gains. When the redundancy is low (0 or 1), and the number of mapping axioms is large, the join selectivity plays an important role in the performance gain, as discussed in [3]; in other cases, the impacts are non-significant.

Figures 5 and 6 report the cardinalities estimated by PostgreSQL divided by the actual sizes of the query answers for all UCQ and JUCQ queries. For UCQs, it shows that PostgreSQL normally underestimates the cardinalities, but it overestimates them when the redundancies are high. As for JUCQS, PostgreSQL always overestimates the cardinalities, ranging from 40 to 200K times. These numbers partially explains why PostgreSQL estimate the costs of both UCQs and JUCQs so badly in Figure 3b.

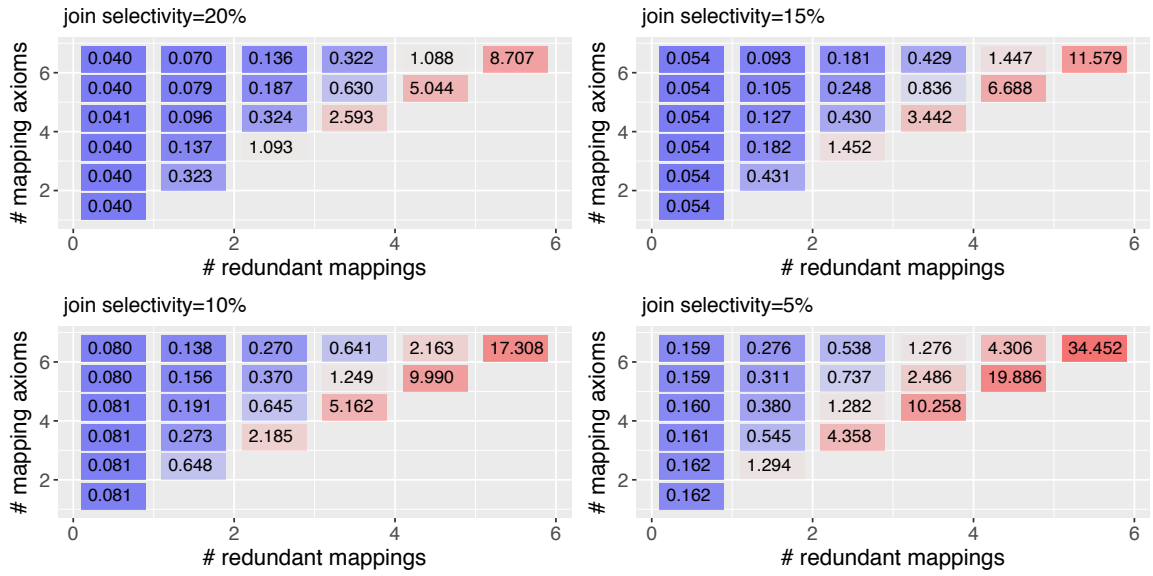


Figure 5: UCQs: (PostgreSQL estimated cardinality) / (real cardinality)

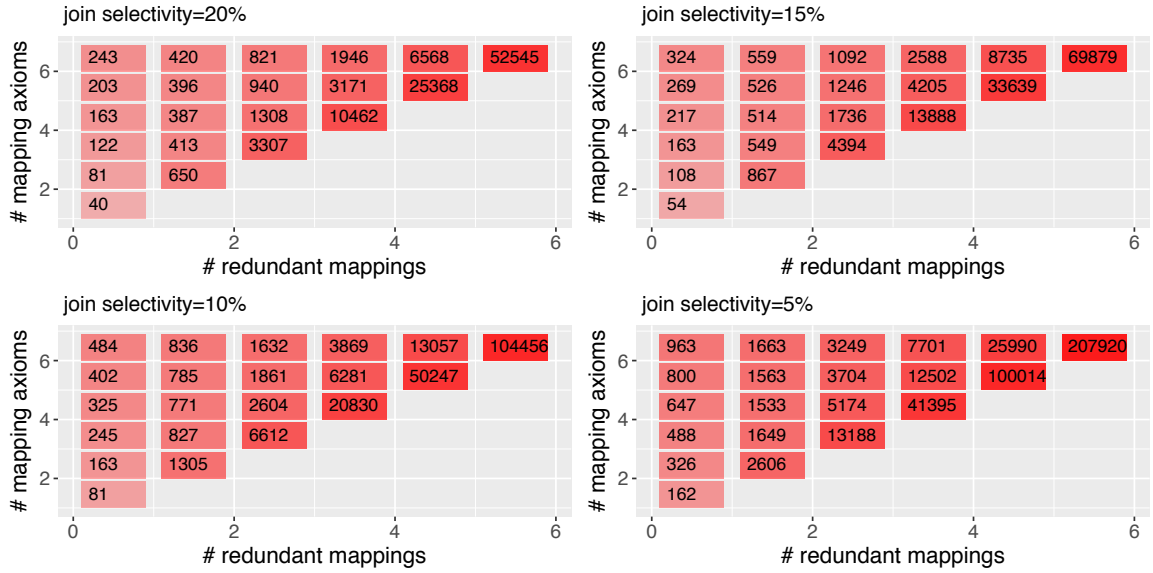


Figure 6: JUCQs: (PostgreSQL estimated cardinality) / (real cardinality)

We obtained similar conclusions for a query with four atoms, and a cover of three fragments.

7.1.1 Wisconsin Experiment II: Four Atoms

In this experiment we consider 84 queries, instances of the following template

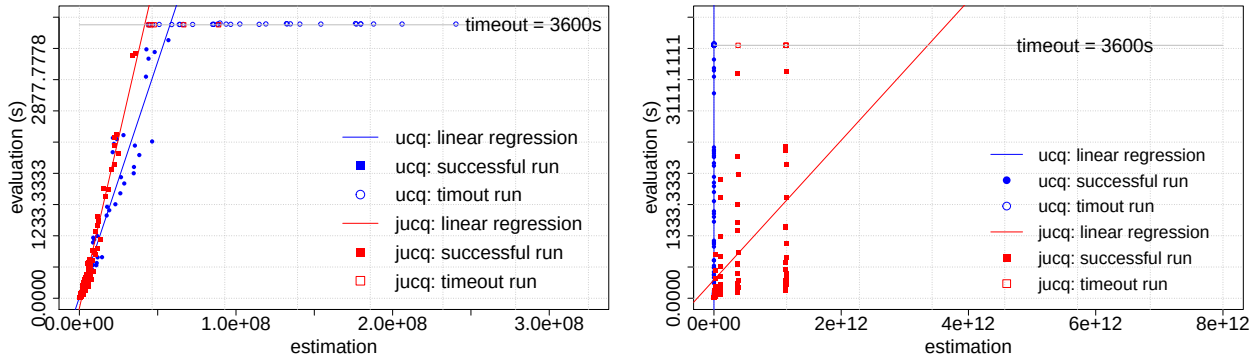
```
SELECT DISTINCT * WHERE {?x :MmRrProp1 ?y1; :JjMmRrProp2 ?y2; :JjMmRrProp3 ?y3; :JjMmRrProp3 ?y4}
```

where $j \in$, m, and r are defined as for the case with three atoms.

For each query, we have tested a correspondent cover query of three fragments f_1, f_2, f_3, where each fragment is an instantiation of the following templates:

```
f1: SELECT DISTINCT ?x ?y1 ?y2 WHERE { ?x :MmRrProp1 ?y1; :JjMmRrProp2 ?y2. }
f2: SELECT DISTINCT ?x ?y3 WHERE { ?x a :MmRrProp1; ?x :JjMmRrProp3 ?y3. }
f3: SELECT DISTINCT ?x ?y4 WHERE { ?x a :MmRrProp1; ?x :JjMmRrProp3 ?y4. }
```

Evaluation. Figures 7, 8, and 9 are the counterparts for the Figures of the three atoms case. Observe that the results look extremely similar, thus confirming our comments from the previous Subsection.



(a) Our cost model vs. evaluation

(b) PostgreSQL cost model vs. evaluation

	queries		linear regression of our cost		linear regression of PostgreSQL cost	
	succ.	time out	b_0	b_1	b_0	b_1
UCQ	50	34	1.22e+01	6.20e-05	8.78e+01	2.21e-05
JUCQ	79	5	-1.30+e02	8.95e-05	2.74e+02	5.84e-10

(c) Results of linear regressions ($evaluation = b_0 + b_1 \times estimation$)

Figure 7: Cost estimations vs evaluation running times

Table 1: Evaluation over the NPD benchmark

SPARQL Query		Unfolding of UCQs		Unfolding of JUCQs		
name	# triple patterns	time (s)	# CQs	time (s)	# Frags	# CQs
q_6	7	2.18	48	1.20	2	14
q_{11}	8	3.39	24	0.40	2	12
q_{12}	10	6.67	48	0.47	2	14
q_{31}	10	54.27	3840	1.58	2	327

7.2 NPD Experiment

The goal of this experiment is to verify that cost-based techniques can improve the performance of query answering over real-world queries and instances. This test is carried on the original real-world instance (as opposed to the scaled data instances) of the NPD benchmark [17] for OBDA systems. We pick the three most challenging UCQ queries (namely q_6, q_{11}, and q_{12}) from the query catalog, and create another even more difficult query (called q_{31}) by combining q_6 and q_9. Query q_{31}, in the listing below, retrieves information regarding wellbores (from q_6) and their related facilities (from q_9).

```
SELECT DISTINCT ?fn ?c ?id ?wlb ?year ?comp WHERE {
  ## Fragment 1
  ?f a npdv:Facility ; npdv:name ?fn ; npdv:regInCountry ?c; npdv:idNPD ?id .
  ?w npdv:productionFacility ?f.

  ## Fragment 2
  ?w rdf:type npdv:Wellbore ; npdv:name ?wlb ; npdv:wellbCYear ?year;
     npdv:drillOpComp  [ npdv:name ?comp ]
FILTER (?id > "0"^^xsd:integer) }
```

In Table 1, we show the evaluation results over the NPD benchmark for UCQs and JUCQs. The unfoldings of JUCQs are constructed using cover queries of 2 fragments, each guided by our cost model. We observe that the sizes of the unfoldings of JUCQs, measured in number of CQs, are sensibly smaller than the size of the unfoldings of UCQs. Finally, we observe that the unfoldings of the JUCQ version of the considered queries improve the running times up to a factor of 34.

8 Conclusion and Future Work

In this paper, we have studied the problem of finding efficient alternative translations of a user query in OBDA. Specifically, we introduced a translation of JUCQ queries that preserves the JUCQ structure while maintaining the possibility of performing joins over database values, rather than URIs constructed by applying mappings definitions. We devised a cost model based on a novel cardinality estimation, for estimating the cost of evaluating a translation of a UCQ or JUCQ over the database. We compared different translations on both a synthetic and fully customizable scenario based on the Wisconsin Benchmark and on a real-world scenario from the NPD Benchmark. In these experiments we have observed

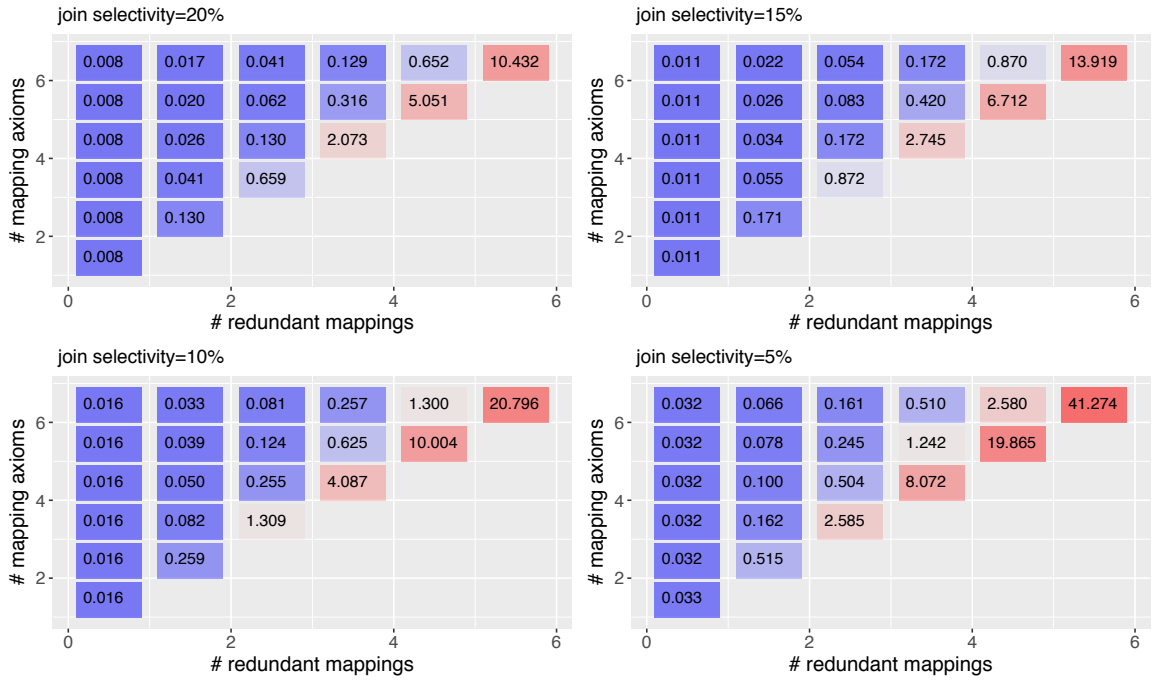


Figure 8: UCQs: (PostgreSQL estimated cardinality) / (real cardinality)

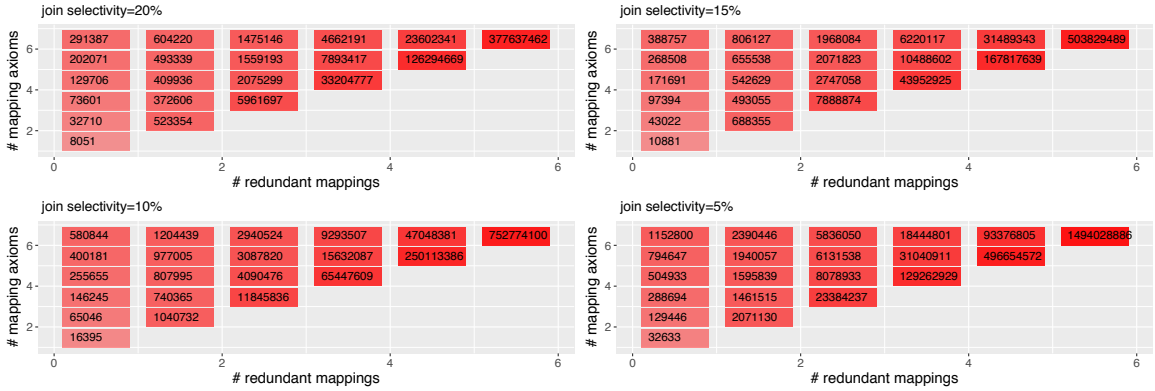


Figure 9: JUCQs: (PostgreSQL estimated cardinality) / (real cardinality)

that (i) our approach based on JUCQ queries can produce translations that are orders of magnitude more efficient than traditional translations into UCQs, and that (ii) the cost model we devised is faithful to the actual query evaluation cost, and hence is well suited to select the best translation.

As future work, we plan to implement our techniques in the state-of-the-art OBDA system *Ontop* and to integrate them with existing optimization strategies. This will allow us to test our approach in more and diversified settings. We also plan to explore alternatives beyond JUCQs. An interesting line of research would be on the problem of relaxing the uniformity assumption made in our cost estimator, by integrating our model with existing techniques based on histograms. Finally, we plan to improve our approach for dealing with projection in an unfolding. In fact, the approach proposed here relies on a formula that has size exponential in the number of queries in the unfolding. To overcome this problem, we plan to remove such formula from our estimation by relying on additional statistics provided by the mapping and the ontology, such as the cardinality of the projection over one of the two arguments of a property defined in the wrap of a T-mapping.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.

[2] Meghyn Bienvenu, Magdalena Ortiz, Mantas Simkus, and Guohui Xiao. Tractable queries for lightweight description logics. In *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI)*. IJCAI/AAAI, 2013.

[3] Damian Bursztyn, François Goasdoué, and Ioana Manolescu. Efficient query answering in DL-Lite through FOL reformulation (extended abstract). In *Proc. of the Int. Workshop on Description Logic (DL)*, volume 1350 of *CEUR Workshop Proceedings,* http://ceur-ws.org/. CEUR-WS.org, 2015.

[4] Damian Bursztyn, François Goasdoué, and Ioana Manolescu. Reformulation-based query answering in RDF: alternatives and performance. *Proc. of the VLDB Endowment*, 8(12):1888–1891, 2015.

[5] Damian Bursztyn, François Goasdoué, and Ioana Manolescu. Teaching an RDBMS about ontological constraints. *Proc. of the VLDB Endowment*, 9(12):1161–1172, 2016.

[6] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web J.*, 8(3):471–487, 2017.

[7] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[8] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF mapping language. W3C Recommendation, World Wide Web Consortium, September 2012. Available at http://www.w3.org/TR/r2rml/.

[9] David J. DeWitt. The Wisconsin benchmark: Past, present, and future. In *The Benchmark Handbook for Database and Transaction Systems*. Morgan Kaufmann, 2 edition, 1993.

[10] Floriana Di Pinto, Domenico Lembo, Maurizio Lenzerini, Riccardo Mancini, Antonella Poggi, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Optimizing query rewriting in ontology-based data access. In *Proc. of the 16th Int. Conf. on Extending Database Technology (EDBT)*, pages 561–572. ACM Press, 2013.

[11] Georges Gardarin, Fei Sha, and Zhao-Hui Tang. Calibrating the query optimizer cost model of iro-db, an object-oriented federated database system. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 378–389, 1996.

[12] Georg Gottlob, Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, Thomas Schwentick, and Michael Zakharyaschev. The price of query rewriting in ontology-based data access. *Artificial Intelligence*, 213:42–59, 2014.

[13] Steve Harris and Andy Seaborne. SPARQL 1.1 query language. W3C Recommendation, World Wide Web Consortium, March 2013. Available at http://www.w3.org/TR/sparql11-query.

[14] Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, and Michael Zakharyaschev. Exponential lower bounds and separation for query rewriting. In *Proc. of the 39th Int. Coll. on Automata, Languages and Programming (ICALP)*, volume 7392 of *Lecture Notes in Computer Science*, pages 263–274. Springer, 2012.

[15] Stanislav Kikot, Roman Kontchakov, and Michael Zakharyaschev. Conjunctive query answering with OWL 2 QL. In *Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*, pages 275–285, 2012.

[16] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and abstract syntax. W3C Recommendation, World Wide Web Consortium, February 2004. Available at http://www.w3.org/TR/rdf-concepts/.

[17] Davide Lanti, Martin Rezk, Guohui Xiao, and Diego Calvanese. The NPD benchmark: Reality check for OBDA systems. In *Proc. of the 18th Int. Conf. on Extending Database Technology (EDBT)*, pages 617–628. OpenProceedings.org, 2015.

[18] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language profiles (second edition). W3C Recommendation, World Wide Web Consortium, December 2012. Available at http://www.w3.org/TR/owl2-profiles/.

[19] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.

[20] Mariano Rodriguez-Muro and Diego Calvanese. High performance query answering over *DL-Lite* ontologies. In *Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*, pages 308–318, 2012.

[21] Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. Ontology-based data access: Ontop of databases. In *Proc. of the 12th Int. Semantic Web Conf. (ISWC)*, volume 8218 of *Lecture Notes in Computer Science*, pages 558–573. Springer, 2013.

[22] Mariano Rodriguez-Muro and Martin Rezk. Efficient SPARQL-to-SQL with R2RML mappings. *J. of Web Semantics*, 33:141–169, 2015.

[23] Juan F. Sequeda and Daniel P. Miranker. Ultrawrap: SPARQL execution on relational data. *J. of Web Semantics*, 22:19–39, 2013.

[24] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts, 5th Edition*. McGraw-Hill Book Company, 2005.

[25] Arun Swami and K. Bernhard Schiefer. On the estimation of join result sizes. In *Proc. of the 4th Int. Conf. on Extending Database Technology (EDBT)*, volume 779 of *Lecture Notes in Computer Science*, pages 287–300. Springer, 1994.