Cost-Driven Ontology-Based Data Access

Davide Lanti, Guohui Xiao(✉), and Diego Calvanese

KRDB Research Centre for Knowledge and Data,
Free University of Bozen-Bolzano, Bolzano, Italy
{dlanti,xiao,calvanese}@inf.unibz.it

Abstract. SPARQL query answering in ontology-based data access (OBDA) is carried out by translating into SQL queries over the data source. Standard translation techniques 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

The paradigm of Ontology-based Data Access (OBDA) [17] presents to the end-users a convenient *virtual RDF graph* [13] view of the data stored in a relational database. Such RDF graph is realized by means of the *TBox of an OWL2 QL ontology* [16] connected to the data source through declarative *mappings* [7]. SPARQL query answering [10] over the RDF graph 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 an SQL query over the data.

In state-of-the-art OBDA systems [5], 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

© Springer International Publishing AG 2017
C. d'Amato et al. (Eds.): ISWC 2017, Part I, LNCS 10587, pp. 452–470, 2017.
DOI: 10.1007/978-3-319-68288-4_27

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 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 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 for 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* [8], a widely adopted benchmark for databases, so as to understand the trade-off between a translation for 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 [14]. 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 provides our characterization for SQL translations of JUCQs. Section 4 presents our novel model for cardinality estimation, and Sect. 5 the associated cost model. Section 6 provides the evaluation of the cost model on the Wisconsin and NPD Benchmarks. Section 7 concludes the paper. Due to space limitation, more details of the techniques, proofs and experiments are provided in an online report [15]. The materials to reproduce the experiments are available online (https://github.com/ontop/ontop-examples/tree/master/iswc-2017-cost).

2 Preliminaries

In this work, we use the **bold** font to denote tuples (when convenient we might treat tuples as sets). Given a tuple of function symbols $\mathbf{f} = (f_1, \ldots, f_n)$ and of variables $\mathbf{x}$, we denote by $\mathbf{f}(\mathbf{x})$ a tuple of terms of the form $(f_1(\mathbf{x_1}), \ldots, f_n(\mathbf{x_n}))$, with $\mathbf{x_i} \subseteq \mathbf{x}$, $1 \leq i \leq n$. We assume some familiarity with basic notions from probability calculus and statistics. We rely on the OBDA framework of [17], 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.

We assume that ontologies are formulated in *DL-Lite*$_\mathcal{R}$ [6], which is the DL providing the formal foundations for OWL 2 QL, the W3C standard ontology language for OBDA [16]. 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 $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ 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 $\mathcal{O}$, we use $cert(q, \mathcal{O})$ to denote the *certain answers* of q over $\mathcal{O}$, which are defined as the set of tuples $\mathbf{a}$ of individuals such that $\mathcal{O} \models q(\mathbf{a})$ (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. A *mapping* m is an expression of the form $L(\mathbf{f}(\mathbf{x})) \leadsto q_m(\mathbf{x})$: the *target part* $L(\mathbf{f}(\mathbf{x}))$ of 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 is a FO query with output variables[2] $\mathbf{x}$. We say that

[1] For conciseness, we use here abstract function symbols in the mapping target. We remind that in concrete mapping languages, such as R2RML [7], 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.

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 [9], we split each mapping $m = L(\mathbf{f}(\mathbf{x})) \leadsto q_m(\mathbf{x})$ 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 $V_m(\mathbf{x}) \leadsto q_m(\mathbf{x})$, and a *high-level* mapping of the form $L(\mathbf{f}(\mathbf{x})) \leadsto V_m(\mathbf{x})$. 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.

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 $\mathcal{A}_{(\mathcal{M},\mathcal{D})} = \{L(\mathbf{f}(\mathbf{a})) \mid L(\mathbf{f}(\mathbf{x})) \leadsto V(\mathbf{x}) \in \mathcal{M} \text{ and } \mathbf{a} \in V(\mathbf{x})^{\mathcal{D}}\}$ 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 $\mathcal{A}$ [6].
- 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 [17].

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., [6,11]. As for the translation, [17] 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.

Definition 1 (Unfolding of a UCQ *[17]***).** *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 [17] that, for a UCQ q, $unf(q, \mathcal{M})$ is an $\mathcal{M}$-translation.

3 Cover-Based Translation in OBDA

We first introduce some terminology from [3], that we use in our technical development. Let q be a query consisting of atoms $\mathcal{F} = \{L_1, \ldots, L_n\}$. A *cover for q* is a collection $C = \{f_1, \ldots, f_m\}$ of non-empty subsets of $\mathcal{F}$, called *fragments*, such that *(i)* $\bigcup_{f_i \in C} f_i = \mathcal{F}$ and *(ii)* no fragment is included into another one. Given a cover C for a query $q(\mathbf{x})$, 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 $f' \neq f$. Consider the query $q_C(\mathbf{x}) \leftarrow \bigwedge_{f \in C} q^{\text{ucq}}_{|_f}(\mathbf{x}_f)$, where $q^{\text{ucq}}_{|_f}(\mathbf{x}_f)$, for each $f \in C$, is a CQ-to-UCQ perfect rewriting of the query $q_{|f}$ w.r.t. $\mathcal{T}$. Then q_C *is a cover-based JUCQ perfect rewriting of q w.r.t. $\mathcal{T}$ and C*, if it is a perfect rewriting of q w.r.t. $\mathcal{T}$.

Authors in [3] have shown that, in *DL-Lite*$_{\mathcal{R}}$, not every cover leads to a cover-based perfect rewriting. Thus, they introduced the notion of *safe covers*, which are covers that guarantee the existence of a cover-based perfect rewriting.

For the remaining part of the section, we fix a query $q(\mathbf{x})$ and a (safe) cover C for it, as well as its cover-based JUCQ perfect rewriting $q_C(\mathbf{x}) \leftarrow \bigwedge_{f \in C} q^{\text{ucq}}_{|f}$ w.r.t $\mathcal{T}$ 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 2 (Unfolding of a JUCQ 1). *For each $f \in C$, let Aux_f be an auxiliary predicate for $q^{ucq}_{|_f}(\mathbf{x}_f)$, and let U_f be a view symbol for the unfolding $unf(q^{ucq}_{|_f}(\mathbf{x}_f), \mathcal{M})$, for each $f \in C$. Consider the set of mappings $\mathcal{M}^{aux} = \{Aux_f(\mathbf{x}_f) \rightsquigarrow U_f(\mathbf{x}_f) \mid f \in C\}$ 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 $unf(q^{aux}_C(\mathbf{x}) \leftarrow \bigwedge_{f \in C} Aux_f(\mathbf{x}_f), \mathcal{M}^{aux})$.*

Theorem 1 (Translation 1). *The query $unf(q_C, \mathcal{M})$ is an $\mathcal{M}$-translation for q_C.*

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, rather than over (indexed) database values themselves (see [15]). In general, such joins cannot be evaluated efficiently by RDBMSs [19]. We introduce a second, less trivial, unfolding characterization that guarantees that joins are performed only over database values. For this we first need to introduce a number of auxiliary notions and results.

Definition 3. *Let* $(L, \mathbf{f}) \in sign(\mathcal{M})$ *be a signature in* $\mathcal{M}$*. Then, the* restriction $\mathcal{M}|_{(L,\mathbf{f})}$ of $\mathcal{M}$ w.r.t. the signature $(L, \mathbf{f})$ *is the set of mappings* $\mathcal{M}|_{(L,\mathbf{f})} = \{m \in \mathcal{M} \mid m = L(\mathbf{f}(\mathbf{v})) \rightsquigarrow V(\mathbf{v})\}$.

Definition 4 (Wrap). *Let* $\mathcal{M}|_{(L,\mathbf{f})} = \{L(\mathbf{f}(\mathbf{v}_i)) \rightsquigarrow V_i(\mathbf{v}_i) \mid 1 \leq i \leq n\}$ *be the restriction of* $\mathcal{M}$ *w.r.t. the signature* $(L, \mathbf{f})$*, and* $\mathbf{f}(\mathbf{v})$ *be a tuple of terms over fresh variables* $\mathbf{v}$*. Then, the* wrap of $\mathcal{M}|_{(L,\mathbf{f})}$ *is the (singleton) set of mappings* $wrap(\mathcal{M}|_{(L,\mathbf{f})}) = \{L(\mathbf{f}(\mathbf{v})) \rightsquigarrow W(\mathbf{v})\}$ *where* W *is a fresh view name for the Datalog query* $(W(\mathbf{v}), \{W(\mathbf{v}_i) \leftarrow V_i(\mathbf{v_i}) \mid 1 \leq i \leq n\})$.

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.*

The wrap operation groups the mappings for a signature into a single mapping. We now introduce an operation that *splits* a mapping according to the function symbols adopted on its source part.

Definition 5 (Split). *Let* $m = L(\mathbf{x}) \rightsquigarrow U(\mathbf{x})$ *be a mapping where* U *is the name for the 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. We denote by* $split(\mathcal{M})$ *the split of the set* $\mathcal{M}$ *of mappings.*

Definition 6 (Unfolding of a JUCQ 2). *Let* q_C^{aux} *be a query and* $\mathcal{M}^{aux}$ *a set of mappings as in Definition 2. Then, the* optimized unfolding $unf_{opt}(q_C(\mathbf{x}), \mathcal{M})$ *of* q_C *w.r.t.* $\mathcal{M}$ *is defined as* $unf(q_C^{aux}(\mathbf{x}), wrap(split(\mathcal{M}^{aux})))$.

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

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 these.

Example 1. 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

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

Translation I. According to Definition 2, 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

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

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

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

Translation II. By Definition 5, 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, 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

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

Finally, by Definition 6 we compute the optimized unfolding of q_C w.r.t. $\mathcal{M}$:

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

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. ■

4 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.

Moreover, in the following, we consider only the restricted form 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 of the form $V^j_{[\cdot]}.\mathbf{x} = V^i_{[\cdot]}.\mathbf{y}$, for some $j < i$. Arbitrary CQs, allowing for projections and arbitrary joins, are considered in the extended version of this work [15].

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*), 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.

Background on Cardinality Estimation. We start by recalling some assumptions that are commonly made by models of cardinality estimation proposed in the database literature (e.g., see [20]): *(i)* For each relation column C, values are *uniformly distributed across* C; intuitively, for a column C of integers, $P(C < v) = (v - \min(C))/(\max(C) - \min(C))$, for each value $v \in C$. *(ii)* There is a *uniform distribution across distinct values*, i.e., $P(C = v_1) = P(C = v_2)$, for all values $v_1, v_2 \in C$. *(iii)* The distributions in different colums are independent, i.e., $P(C_1 = v_1 | C_2 = v_2) = P(C_1 = v_1)$, for all values $v_1 \in C_1$ and $v_2 \in C_2$. *(iv)* Columns in a join condition match "as much as possible", i.e., given a join $V \bowtie_{\mathbf{x}=\mathbf{y}} W$, it is assumed that $|\pi_{\mathbf{x}}(V^{\mathcal{D}}) \Cap \pi_{\mathbf{y}}(W^{\mathcal{D}})| = \min(|\pi_{\mathbf{x}}(V)|, |\pi_{\mathbf{y}}(W)|)$.

Given the assumptions, the cardinality of a join $V \bowtie_{\mathbf{x}=\mathbf{y}} W$ is estimated [21] as:

$$k_{\mathcal{D}}(V \bowtie_{\mathbf{x}=\mathbf{y}} W) \cdot |V^{\mathcal{D}}|/\mathit{dist}_{\mathcal{D}}(V, \mathbf{x}) \cdot |W^{\mathcal{D}}|/\mathit{dist}_{\mathcal{D}}(W, \mathbf{y}) \tag{1}$$

where $k_{\mathcal{D}}$ is an estimation of the number of distinct values satisfying the join condition (i.e., $k_{\mathcal{D}}$ estimates $|\pi_{\mathbf{x}}(V^{\mathcal{D}}) \Cap \pi_{\mathbf{y}}(W^{\mathcal{D}})|$, and $\mathit{dist}_{\mathcal{D}}(V, \mathbf{x})$ (resp., $\mathit{dist}_{\mathcal{D}}(W, \mathbf{y})$) 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.

Of the assumptions *(i)*–*(iv)* above, we maintain only *(ii)* and *(iii)* in our cardinality estimator, while we drop *(i)* and *(iv)* due to the additional information given by the structure of the mappings. In the following, we will show how even under these conditions we can use Formula (1), to estimate the cardinality of conjunctive queries.

Basic CQ Cardinality Estimation. We first generalize Formula (1) to basic CQs.

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

$$f_{\mathcal{D}}(E \bowtie_{V_{[p]}.\mathbf{x}=W_{[q]}.\mathbf{y}} W_{[q]}) = \begin{cases} \left\lceil \dfrac{k_{\mathcal{D}}(V_{[p]} \bowtie_{V_{[p]}.\mathbf{x}=W_{[q]}.\mathbf{y}} W_{[q]}) \cdot |V^{\mathcal{D}}| \cdot |W^{\mathcal{D}}|}{dist_{\mathcal{D}}(V, V_{[p]}.\mathbf{x}) \cdot dist_{\mathcal{D}}(W, W_{[q]}.\mathbf{y})} \right\rceil, & \text{if } E = V \\ \left\lceil \dfrac{k_{\mathcal{D}}(E \bowtie_{V_{[p]}.\mathbf{x}=W_{[q]}.\mathbf{y}} W_{[q]}) \cdot f_{\mathcal{D}}(E) \cdot |W^{\mathcal{D}}|}{dist_{\mathcal{D}}(E, V_{[p]}.\mathbf{x}) \cdot dist_{\mathcal{D}}(W, V_{[p]}.\mathbf{y})} \right\rceil, & \text{otherwise.} \end{cases} \tag{2}$$

Our cardinality estimator exploits assumptions *(ii)* and *(iii)* above, and relies on our definitions of the *facing values estimator* $k_{\mathcal{D}}$ and of the *distinct values estimator* $dist_{\mathcal{D}}$, which are based on additional statistics collected with the help of the mappings, instead of being based on assumptions *(i)* and *(iv)*, as in Formula (1).

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

$$k_{\mathcal{D}}(E \bowtie_{V_{[p]}.\mathbf{x}=W_{[q]}.\mathbf{y}} W_{[q]}) = \begin{cases} |\pi_{\mathbf{x}}(V^{\mathcal{D}}) \Cap \pi_{\mathbf{y}}(W^{\mathcal{D}})|, & \text{if } E = V \\ \left\lceil |\pi_{\mathbf{x}}(V^{\mathcal{D}}) \Cap \pi_{\mathbf{y}}(W^{\mathcal{D}})| \cdot \dfrac{dist_{\mathcal{D}}(E, V_{[p]}.\mathbf{x})}{dist_{\mathcal{D}}(V, V_{[p]}.\mathbf{x})} \right\rceil, & \text{otherwise,} \end{cases} \tag{3}$$

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 *(ii)*.

Distinct Values Estimator. Let Q be a set of qualified attributes, and E be basic CQ. We define the set $\mathrm{ea}(E, Q)$ of equivalent attributes of Q in E as $\bigcup_{i>0} C_i$, where *(i)* $C_1 := \{Q\}$ *(ii)* $C_{n+1} := C_n \cup \{Q' \mid \exists Q'' \in C_n \text{ s.t. } Q' = Q'' \text{ or } Q'' = Q' \text{ is a join condition in } E\}, n \geq 1$. 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 $U_{[r]}.\mathbf{z} \in \mathrm{ea}(E, V_{[p]}.\mathbf{x})$, 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

$$dist_{\mathcal{D}}(E, V_{[p]}.\mathbf{x}) = \begin{cases} |\pi_{\mathbf{x}}(V^{\mathcal{D}})|, & \text{if } E = V \\ \min\left\{\left\lceil k_{\mathcal{D}}(E') \cdot \dfrac{f_{\mathcal{D}}(E)}{f_{\mathcal{D}}(E')}\right\rceil, k_{\mathcal{D}}(E')\right\}, & \text{if } se(E, V_{[p]}.\mathbf{x}) = E' \neq \bot \\ \min\left\{\left\lceil |\pi_{\mathbf{x}}(V^{\mathcal{D}})| \cdot \dfrac{f_{\mathcal{D}}(E)}{|V^{\mathcal{D}}|}\right\rceil, |\pi_{\mathbf{x}}(V^{\mathcal{D}})|\right\}, & \text{otherwise.} \end{cases} \tag{4}$$

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 $\frac{f_{\mathcal{D}}(E)}{f_{\mathcal{D}}(E')}$ and $\frac{f_{\mathcal{D}}(E)}{|V^{\mathcal{D}}|}$ are again scaling factors relying on assumption *(ii)*. 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.

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	...

Fig. 1. Data instance $\mathcal{D}$.

Example 2. Consider the data instance $\mathcal{D}$ from Fig. 1. Relevant statistics are:

- $|T_1^{\mathcal{D}}| = 5, \quad |T_2^{\mathcal{D}}| = |T_3^{\mathcal{D}}| = 10$
- $|\pi_{\mathrm{a}}(T_1^{\mathcal{D}})| = |\pi_{\mathrm{d}}(T_2^{\mathcal{D}})| = 5, \quad |\pi_{\mathrm{c}}(T_2^{\mathcal{D}})| = |\pi_{\mathrm{f}}(T_3^{\mathcal{D}})| = |\pi_{\mathrm{e}}(T_3^{\mathcal{D}})| = 10,$
- $|\pi_{\mathrm{a}}(T_1^{\mathcal{D}}) \Cap \pi_{\mathrm{c}}(T_2^{\mathcal{D}})| = 3, \quad |\pi_{\mathrm{d}}(T_2^{\mathcal{D}}) \Cap \pi_{\mathrm{e}}(T_3^{\mathcal{D}})| = 5, \quad |\pi_{\mathrm{a}}(T_1^{\mathcal{D}}) \Cap \pi_{\mathrm{f}}(T_3^{\mathcal{D}})| = 1.$

We calculate $f_{\mathcal{D}}(E)$ for the basic CQ $E = T_1 \bowtie_{T_1.\mathrm{a}=T_2.\mathrm{c}} T_2 \bowtie_{T_2.\mathrm{d}=T_3.\mathrm{e}} T_3 \bowtie_{T_1.\mathrm{a}=T_3'.\mathrm{f}} T_3'$, 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 $f_{\mathcal{D}}(E^{(1)})$ and $f_{\mathcal{D}}(E^{(2)})$.

$$\begin{aligned} f_{\mathcal{D}}(E^{(1)}) &= f_{\mathcal{D}}(T_1 \bowtie_{T_1.\mathrm{a}=T_2.\mathrm{c}} T_2) = \left\lceil \frac{k_{\mathcal{D}}(T_1 \bowtie_{T_1.\mathrm{a}=T_2.\mathrm{c}} T_2) \cdot |T_1^{\mathcal{D}}| \cdot |T_2^{\mathcal{D}}|}{dist_{\mathcal{D}}(T_1, \mathrm{a}) \cdot dist_{\mathcal{D}}(T_2, \mathrm{c})} \right\rceil \\ &= \left\lceil \frac{|\pi_{\mathrm{a}}(T_1^{\mathcal{D}}) \Cap \pi_{\mathrm{c}}(T_2^{\mathcal{D}})| \cdot |T_1^{\mathcal{D}}| \cdot |T_2^{\mathcal{D}}|}{|\pi_{\mathrm{a}}(T_1^{\mathcal{D}})| \cdot |\pi_{\mathrm{c}}(T_2^{\mathcal{D}})|} \right\rceil = \lceil (3 \cdot 5 \cdot 10)/(5 \cdot 10) \rceil = 3 \\ f_{\mathcal{D}}(E^{(2)}) &= f_{\mathcal{D}}(E^{(1)} \bowtie_{T_2.\mathrm{d}=T_3.\mathrm{e}} T_3) = \left\lceil \frac{k_{\mathcal{D}}(E^{(1)} \bowtie_{T_2.\mathrm{d}=T_3.\mathrm{e}} T_3) \cdot f_{\mathcal{D}}(E^{(1)}) \cdot |T_3^{\mathcal{D}}|}{dist_{\mathcal{D}}(E^{(1)}, T_2.\mathrm{d}) \cdot dist_{\mathcal{D}}(T_3, \mathrm{e})} \right\rceil \end{aligned} \tag{5}$$

By Formula (4), $dist_{\mathcal{D}}(E^{(1)}, T_2.\mathrm{d})$ in Formula (5) can be calculated as

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

By Formula (3), $k_{\mathcal{D}}(E^{(1)} \bowtie_{T_2.\mathrm{d}=T_3.\mathrm{e}} T_3)$ in Formula (5) can be calculated as

$$\begin{aligned} k_{\mathcal{D}}(E^{(1)} \bowtie_{T_2.\mathrm{d}=T_3.\mathrm{e}} T_3) &= \left\lceil \frac{k_{\mathcal{D}}(T_2 \bowtie_{T_2.\mathrm{d}=T_3.\mathrm{e}} T_3)}{dist_{\mathcal{D}}(T_2, \mathrm{d})} \cdot dist_{\mathcal{D}}(E^{(1)}, T_2.\mathrm{d})\right\rceil \\ &= \left\lceil \frac{|\pi_{\mathrm{d}}(T_2^{\mathcal{D}}) \Cap \pi_{\mathrm{e}}(T_3^{\mathcal{D}})|}{|\pi_{\mathrm{d}}(T_2^{\mathcal{D}})|} \cdot dist_{\mathcal{D}}(E^{(1)}, T_2.\mathrm{d})\right\rceil = \left\lceil \frac{5}{5} \cdot 2\right\rceil = 2 \end{aligned}$$

By plugging the values for $k_{\mathcal{D}}$ and $dist_{\mathcal{D}}$ in Formula (5), we obtain

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

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

$$f_{\mathcal{D}}(E) = f_{\mathcal{D}}(E^{(2)} \bowtie_{T_1.\mathrm{a}=T_3'.\mathrm{f}} T_3') = \left\lceil \frac{k_{\mathcal{D}}(E^{(2)} \bowtie_{T_1.\mathrm{a}=T_3'.\mathrm{f}} T_3') \cdot f_{\mathcal{D}}(E^{(2)}) \cdot |T_3^{\mathcal{D}}|}{dist_{\mathcal{D}}(E^{(2)}, T_1.\mathrm{a}) \cdot dist_{\mathcal{D}}(T_3, \mathrm{f})}\right\rceil \tag{6}$$

By Formula (4), $dist_{\mathcal{D}}(E^{(2)}, T_1.\mathrm{a})$ in Formula (6) can be computed as

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

Then, by Formula (3), $k_{\mathcal{D}}(E^{(2)} \bowtie_{T_1.\mathrm{a}=T_3'.\mathrm{f}} T_3')$ in Formula (6) can be computed as

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

By plugging the values for $k_{\mathcal{D}}$ and $dist_{\mathcal{D}}$ in (6), we finally obtain

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

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

Collecting the Necessary 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 of mappings $\mathcal{M} = \{L_i(\mathbf{f_i}(\mathbf{v_i})) \rightsquigarrow V_i(\mathbf{v_i}) \mid 1 \leq i \leq n\}$ 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 1 contains only joins between pairs of source queries considered by Statistics S_3.

Unfolding Cardinality Estimator. We now show how to estimate the cardinality of an unfolding by using the Formulas (2), (3), and (4) 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 *wrap* mappings, and *(ii)* the query to unfold is a CQ.

Theorem 3. *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*

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

The previous theorem states that the cardinality of the unfolding of a query over a wrap mapping 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 [15] relies on the fact that, when wrap mappings are 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 $\mathit{wrap}(\mathcal{M})$ rather than $\mathcal{M}$, and sum up the estimations for each CQ in $\mathit{unf}(q, \mathit{wrap}(\mathcal{M}))$.

The method above might overestimate the actual cardinality if the input CQ contains non-answer variables. In [15] 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 rewriting is in general a UCQ. This is usually not a critical aspect, especially in practical applications of OBDA. By using saturated (or T-)mappings [18] $\mathcal{M}_{\mathcal{T}}$ in place of $\mathcal{M}$, in fact, the rewriting of an input CQ q

almost always [12] coincides with q itself[3]. Hence, in most cases we can directly use in Theorem 3 the input query q, if we use $wrap(\mathcal{M}_\mathcal{T})$ instead of $wrap(\mathcal{M})$. A fully detailed example on how this is done is provided in [15].

5 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 [20]. We here provide the high-level view of the cost model, and leave the details in [15].

Cost for the Unfolding of a UCQ. Recall from Sect. 3 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

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

where

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

Cost for the Unfolding of a JUCQ. Recall from Sect. 3 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

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

where

- $c(q_i^{ucq})$ 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* [20] and not materialized;
- $c_{mj}(q^{jucq})$ is the cost of a merge join over the materialized intermediate results;
- $c_u(q^{jucq})$ is the cost of removing duplicate 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., Sect. 3). Otherwise, we need to consider the cost of eliminating duplicate results.

[3] *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.

6 Experimental Results

In this section, we provide an empirical evaluation that compares unfoldings for UCQs and (optimized) unfoldings for 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 3.47 GHz), 106 GB of RAM and five 1TB 15K RPM HDs. As RDBMS we have used PostgreSQL 9.6. In the extended version [15] of this work we provide the material to replicate our experiments.

Wisconsin Experiment. This experiment is based on the *Wisconsin Benchmark* [8], which allows for in-detail analyses w.r.t. parameters such as join selectivities. We created several copies of the Wisconsin table, and populated each of them with 1M rows. 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 200 k 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 definining the property, minus one.

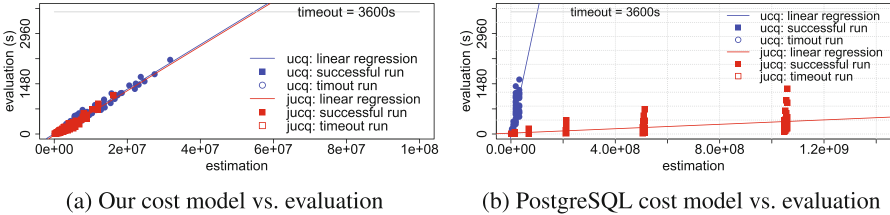
For each query, we have tested a correspondent cover query of two fragments f_1, f_2, where each fragment is an instantiation of the following templates:

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

We have implemented our cost model in a Python script. For each SPARQL query, we compute the estimation of the costs of both unfoldings for UCQs and JUCQs, and evaluate these unfoldings over the PostgreSQL database with a timeout of 20 min.

In Fig. 2, 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 s and 350 s.
- In Fig. 2a, 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.



(a) Our cost model vs. evaluation (b) PostgreSQL cost model vs. evaluation

	queries		linear regression of our cost estimation		linear regression of PostgreSQL cost estimation	
	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 ($evaluation = b_0 + b_1 \times estimation$)

Fig. 2. Cost estimations vs evaluation running times

– Fig. 2b 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 Fig. 3, we visualize the performance gain of JUCQs compared with UCQs. The four subgraphs correspond to four different 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 framed-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.

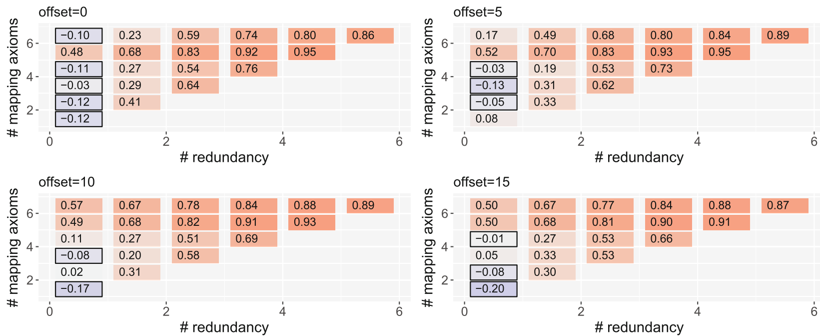


Fig. 3. Performance gain of JUCQ compared with UCQ

Figures 4 and 5 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 200 K times. These numbers partially explain why PostgreSQL estimate the costs of both UCQs and JUCQs so badly in Fig. 2b.

We obtained similar conclusions for a query with four atoms, and a cover of three fragments. For more details, refer to the extended version [15] of this work.

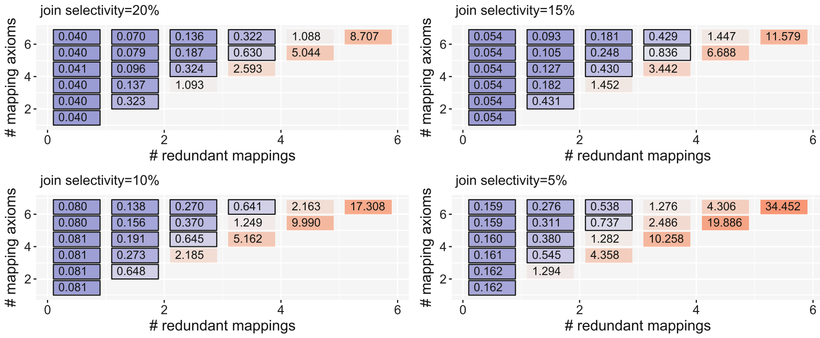


Fig. 4. UCQs: (PostgreSQL estimated cardinality)/(real cardinality)

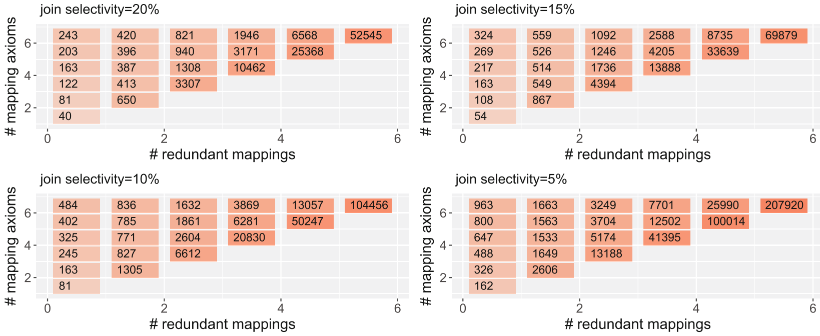


Fig. 5. JUCQs: (PostgreSQL estimated cardinality)/(real cardinality)

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 [14] for OBDA systems. We pick the three most challenging UCQ queries (namely q_6, q_{11}, q_{12},

Table 1. Evaluation over the NPD benchmark

SPARQL query		Unfolding for UCQs		Unfolding for 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

and q_{31}) from the query catalog, where q_{31} is a combination of queries q_6 and q_9, created during this work, which retrieves information regarding wellbores (from q_6) and their related facilities (from q_9).

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

7 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 for 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 for 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 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. Finally, we plan to work on the problem of relaxing the uniformity assumption made in our cost estimator, by integrating our model with existing techniques based on histograms.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley Publishing Co., Boston (1995)
2. Bienvenu, M., Ortiz, M., Simkus, M., Xiao, G.: Tractable queries for lightweight description logics. In: Proceedings of IJCAI, IJCAI/AAAI (2013)
3. Bursztyn, D., Goasdoué, F., Manolescu, I.: Efficient query answering in DL-Lite through FOL reformulation (extended abstract). In: Proceedings of DL, vol. 1350. CEUR, CEUR-WS.org (2015). http://ceur-ws.org/Vol-1350/paper-15.pdf
4. Bursztyn, D., Goasdoué, F., Manolescu, I.: Reformulation-based query answering in RDF: alternatives and performance. PVLDB **8**(12), 1888–1891 (2015). http://www.vldb.org/pvldb/vol8/p1888-bursztyn.pdf
5. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: answering SPARQL queries over relational databases. Semant. Web J. **8**(3), 471–487 (2017). http://dx.doi.org/10.3233/SW-160217
6. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: the DL-Lite family. JAR **39**(3), 385–429 (2007)
7. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language. W3C Recommendation, W3C, September 2012. http://www.w3.org/TR/r2rml/
8. DeWitt, D.J.: The Wisconsin benchmark: past, present, and future. In: Gray, J. (ed.) The Benchmark Handbook for Database and Transaction Systems, 2nd edn. Morgan Kaufmann, San Mateo (1993)
9. Di Pinto, F., Lembo, D., Lenzerini, M., Mancini, R., Poggi, A., Rosati, R., Ruzzi, M., Savo, D.F.: Optimizing query rewriting in ontology-based data access. In: Proceedings of EDBT, pp. 561–572. ACM Press and Addison Wesley (2013)
10. Harris, S., Seaborne, A.: SPARQL 1.1 query language. W3C Recommendation, W3C, March 2013. http://www.w3.org/TR/sparql11-query
11. Kikot, S., Kontchakov, R., Podolskii, V., Zakharyaschev, M.: Exponential lower bounds and separation for query rewriting. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012. LNCS, vol. 7392, pp. 263–274. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31585-5_26
12. Kikot, S., Kontchakov, R., Zakharyaschev, M.: Conjunctive query answering with OWL 2 QL. In: Proceedings of KR, pp. 275–285 (2012)
13. Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): concepts and abstract syntax. W3C Recommendation, W3C, February 2004. http://www.w3.org/TR/rdf-concepts/
14. Lanti, D., Rezk, M., Xiao, G., Calvanese, D.: The NPD benchmark: reality check for OBDA systems. In: Proceedings of EDBT, pp. 617–628 (2015). http://openproceedings.org/
15. Lanti, D., Xiao, G., Calvanese, D.: Cost-driven ontology-based data access (extended version). CoRR abs/1707.06974 (2017). http://arxiv.org/abs/1707.06974
16. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language Profiles, 2nd edn. W3C Recommendation, W3C, December 2012. http://www.w3.org/TR/owl2-profiles/
17. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. In: Spaccapietra, S. (ed.) Journal on Data Semantics X. LNCS, vol. 4900, pp. 133–173. Springer, Heidelberg (2008). doi:10.1007/978-3-540-77688-8_5

18. Rodriguez-Muro, M., Calvanese, D.: High performance query answering over DL-Lite ontologies. In: Proceedings of KR, pp. 308–318 (2012)
19. Rodriguez-Muro, M., Rezk, M.: Efficient SPARQL-to-SQL with R2RML mappings. J. Web Semant. **33**, 141–169 (2015)
20. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database System Concepts, 5th edn. McGraw-Hill Book Company, Boston (2005)
21. Swami, A., Schiefer, K.B.: On the estimation of join result sizes. In: Jarke, M., Bubenko, J., Jeffery, K. (eds.) EDBT 1994. LNCS, vol. 779, pp. 287–300. Springer, Heidelberg (1994). doi:10.1007/3-540-57818-8_58