Graph-driven Federated Data Management

Sergi Nadal, Alberto Abelló, Oscar Romero, Stijn Vansummeren, Panos Vassiliadis, *Senior Member, IEEE*

Abstract—Modern data analysis applications require the ability to provide on-demand integration of data sources while offering a flexible and user-friendly query interface. Traditional techniques for answering queries using views, focused on a rather static setting, fail to address such requirements. To overcome these issues, we propose a fully-fledged data integration approach based on graph-based constructs. The extensibility of graphs allows us to extend the traditional framework for data integration with view definitions. Furthermore, we also propose a query language based on subgraphs. We tackle query answering via a query rewriting algorithm based on well-known algorithms for answering queries using views. We experimentally show that the proposed method yields good performance and does not introduce a significant overhead.

Index Terms—Data integration, data wrangling, GLAV mappings.

1 Introduction

Data wrangling is defined as an iterative data exploration process to enable analysis [1]. In contrast to *data warehousing* approaches, where data are materialized in a target schema tailored to a specific kind of analysis, *virtual data integration* systems play a key role on the exploration of a wealth of data that is yet to be integrated [2]. Traditionally, virtual integration systems have aimed to expose a single mediated relational schema. Alternatively, given that graphs are widely accepted as a convenient data model to represent real-world abstractions and their relationships [3], the community has proposed different solutions grounded on this formalism. However, as the popularity of wrangling systems grows, non-technical users face high-entry barriers on interacting with them, requiring queries to be written in technical languages such as Datalog [4] or SPARQL [5]. Additionally, the vast number of available heterogeneous and independent datasets on the web pose several challenges for contemporary wrangling demands [6]. Hence, the development of **flexible** and **easy-to-use** data integration systems remains an open research topic [7], [8].

We distinguish virtual integration proposals according to *(a)* the data model/query language, and *(b)* the kind of mapping used to connect the sources and the global schema. A summary of the main approaches is presented in Table 1. Regarding data model and query language, we identify the classical (relational) database (**DB**) approach, and the knowledge representation (**KR**) one. The former, aims to expose a single relational schema as the integrated database, while the latter relies on well-behaved fragments of description logics to reason about data and incorporate new facts [9]. Regarding mappings, we have: *global-as-view* (**GAV**) characterizing the target schema in terms of queries over the sources; *local-as-view* (**LAV**) characterizing sources in terms of queries over the target schema; and the most generic *global-local-as-view* (**GLAV**) characterizing queries over the sources in terms of queries over the target schema.

	GAV	LAV	GLAV
DB	[10], [11], [12]	[13],[14],[15]	[16],[17]
KR	[5], [18], [19]	[20],[21], [22]	[4], [23]

TABLE 1: Overview of approaches w.r.t. data model and query language (rows) and kind of mappings (columns)

Nonetheless, DB-based integration systems require users to explicitly state shared join variables in, commonly conjunctive, queries (e.g., a Datalog query like $R(x, y), S(y, z)$). This requires an accurate understanding of the schema in-use as well as the query language. Alternatively, KR-based systems expose a graph-based data model enabling expressive visual query paradigms to non-expert users [24]. KR-based systems adopting GAV mappings are limited by the management of evolution in the sources (i.e., adding or modifying the structure of a source might require revisiting multiple mapping definitions). This limitation is lifted by LAV/GLAV-based systems. However, these approaches are inherently more complex than those in the DB category due to the embedded reasoning capabilities. To this end, the goal of this paper is to provide a single, coherent, all-encompassing virtual data integration model of *(a)* schemata, and *(b)* queries, in a way that facilitates *(i)* easy registration of sources under a global schema, *(ii)* easy (visual) query formulation without the need for expressing joins, *(iii)* automatic translation of queries over the global schema to queries over the sources at runtime, and *(iv)* absence of any reasoning mechanism. In order to contribute towards this goal, we build and extend previous work [25], where we presented an ontology for query answering over linear and acyclic queries. Here, we present a novel and fully-fledged approach to virtual integration using graphs as canonical data model for the whole process. Precisely, we present a framework for query answering over graphs mediating a set of heterogeneous data sources connected

- *Sergi Nadal, Alberto Abelló, and Oscar Romero, Universitat Politècnica de Catalunya, Barcelona, Spain.*
  E-mail: {snadal,aabello,oromero}@essi.upc.edu
- *Stijn Vansummeren, UHasselt - Hasselt University, Data Science Institute, Agoralaan, 3590 Diepenbeek, Belgium.*
  E-mail: stijn.vansummeren@uhasselt.be
- *Panos Vassiliadis. University of Ioannina, Ioannina, Greece.*
  E-mail: pvassil@cs.uoi.gr

Manuscript received ?; revised ??

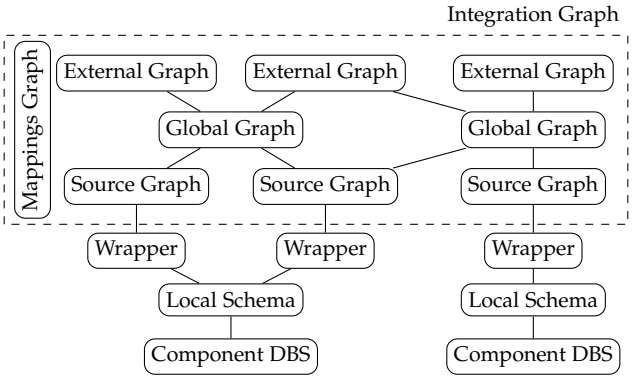


Fig. 1: Building blocks in an Integration Graph, adapted from Sheth & Larson (1990)

via GLAV mappings. The proposed graph-based framework (i.e., the *integration graph*) takes as building blocks Seth and Larson's reference architecture for federated database systems [26], and adapts its components to Lenzerini's data integration framework [27]. As depicted in Figure 1, an integration graph contains all the metadata constructs representing a federated system.

The main novelty of our approach is a query language based on coverings, or *contours*, of a graph representing the global schema. The proposed language does not require users to define join conditions, a task delegated to the rewriting algorithm. This, enables a visual representation of the query. Furthermore, as opposed to classic methods, where different data structures are maintained for schemata and mappings, our framework is entirely grounded on graphs as unique data structure for all constructs. Besides the flexibility and ease of use that bring to the wrangling process, using graphs to represent data integration systems brings performance benefits. Encoding all required metadata (i.e., global schema, source descriptions, mappings and queries) in a single data structure simplifies the interoperability among them. This allows rewriting algorithms to query such metadata structures (e.g., mappings), bringing the ability to efficiently identify the relevant sources containing a query posed over the global schema. Our approach is theoretically and experimentally validated, proving its soundness and showing its practical efficiency.

Contributions. We summarize our contributions as follows:

- We propose a novel graph-based framework for virtual data integration with GLAV mappings.
- We introduce the notion of *minimally-sound* and *minimally-complete* rewriting algorithms, which guarantee to yield the *maximally-contained rewritings* of a query.
- We present a query rewriting algorithm, satisfying the above properties, that reformulates graph-based queries into maximally-contained rewritings. A distinguishing feature of our approach is that, by considering the composition of sources, we yield more results than the alternative methods with no performance overhead.

Outline. The rest of the paper is structured as follows. In Sections 2 and 3 we related work and formalize our framework. In Section 4, we present the rewriting algorithm, followed by an analysis of its computational complexity and a theoretical validation in Section 5. In Section 6, we experimentally validate our approach. We finally conclude our paper and present future work in Section 7.

2 Related Work

The problem of answering queries using views has set the theoretical underpinnings for several data integration approaches [28]. As shown in Table 1, we distinguish two families of systems: those exposing a relational schema as integrated database (DB), and those exposing an ontology (KR) as mediator. Here, we review related work for each family distinguishing on the kind of used mappings.

DB-based approaches. GAV-based integration systems such as TSIMMIS [10], Garlic [11] or MOMIS [12] pioneered the field of data integration. With the bloom of web sources the community started paying attention to LAV mappings due to their expressive power for heterogeneous sources. Rewriting LAV mappings is equivalent to the problem of answering queries using views [28]. In data integration, it is common to seek maximally-contained rewritings, being the *bucket algorithm* [13], the *inverse rules algorithm* [14] and the *MiniCon algorithm* [15] the most prominent techniques. Yet, several methods have been proposed paying special attention to the scalability of the rewriting process. [29] proposes to use graphs to represent distinguished and existential variables as intermediate data structure in the rewriting process. This allows to detect common access patterns across sources and generate a compact representation, showing scalability results up to 10.000 views. Extensions of this model have also been proposed focusing on the chase algorithm [30]. Regarding GLAV mappings, these were originally designed for data exchange [16]. Query answering consists on computing the *chase* over instances of the source schema, generating new facts until all dependencies are satisfied. However, the scalability of this method is still a major drawback [31].

KR-based approaches. The *ontology-based data access* (OBDA) approach is the main representative of graph mediation. OBDA systems implement a virtual integration approach using ontologies [32]. To this end, they adopt the *DL-Lite* family of description logics as foundation, a well-behaved fragment capturing a fair portion of conceptual modeling formalisms, and guarantee *first-order rewritability* of queries [9]. The ontology can be leveraged to complement query results with further knowledge via reasoning, thus being able to compute the certain answers. Most approaches adopt GAV mappings, and thus the query answering task is reduced to unfolding mappings [33]. Popular GAV-based OBDA systems are Ontop [5], Morph [18] and Mastro [19]. Besides the popularity of GAV-based OBDA systems, LAV [20],[21],[22] and GLAV [17], [34] mappings have also been studied for description logics. Recently, new approaches to OBDA using GLAV mappings have been proposed to combinedly query the ontology and its instances [23].

As conclusion, we acknowledge that, to the best of our knowledge, there is no work considering the intersection of our problems of interest (i.e., query answering over graphs without reasoning). Hence, our approach is complementary to those presented, and fills a gap in scenarios where users are non-technical and inference is not required.

3 PRELIMINARIES

In this section, we introduce the formal background of our approach, which allow to define the necessary conditions for correct query rewriting algorithms.

3.1 Data source model and queries

Relations and wrappers. A schema R is a finite nonempty set of relational symbols $\{r_1, \ldots, r_m\}$, where each r_i has a fixed arity n_i. Let A be a set of attribute names, then each $r_i \in R$ is associated to a tuple of attributes denoted by $att(r_i)$. Henceforth, we will assume that $\forall i, j : i \neq j \rightarrow att(r_i) \cap att(r_j) = \emptyset$ (i.e., relations do not share attribute names), which can be simply done prefixing attribute names with their relation name. Let D be an infinite set of values, a tuple t in r_i is a function $t : att(r_i) \rightarrow D$. For any relation r_i, $tuples(r_i)$ denotes the set of all possible tuples for r_i. A wrapper w is an element in R with a function $exec(w)$ that returns a set of relational tuples $T \subseteq tuples(w)$. In practice, wrappers can be implemented via any black box program as long as there exists a mapping function from their specific data model to *first normal form* (1NF).

Conjunctive queries. A conjunctive query (CQ) is an expression of the form

$$Q = \pi_{\overline{y}}(w_1 \times \ldots \times w_n) \mid \bigwedge_{i=1}^{m} P_i(\overline{z_i}))$$

where $w_1, \ldots, w_n$ are distinct wrappers; $P_1, \ldots, P_n$ are equi-join predicates respectively over $\overline{z_1}, \ldots \overline{z_n}$; and both $\bigcup_{i=1}^{m} \overline{z_i}$ and $\overline{y}$ are subsets of $\bigcup_{i=1}^{n} att(w_i)$. Throughout the paper, we might refer to a CQ as a 3-tuple $Q = \langle \pi, \bowtie, W \rangle$ respectively denoting the sets of projected attributes, equi-join predicates and wrappers of Q. We might also refer to binary equi-join predicates as pairs of the form $p = \langle a_1, a_2 \rangle$, noting that $\langle a_1, a_2 \rangle = \langle a_2, a_1 \rangle$. Next, we define the functions $att(Q)$, $wrap(Q)$, $pred(Q)$ and $predatt(Q)$ respectively denoting the sets of projected attributes π, wrappers W, equi-join predicates $\bowtie$, and attributes contained in the equi-join predicates of Q. We define the composition of two CQs ($Q = Q_1 \oplus Q_2$) as $Q = \langle att(Q_1) \cup att(Q_2), pred(Q_1) \cup pred(Q_2), wrap(Q_1) \cup wrap(Q_2) \rangle$. Note the presented syntax of CQs does not include filters (e.g., $w_1.age > 30$). Without loss of generality, it is always possible to push down unary selection predicates on top of every wrapper. We use $exec(Q)$ to denote the execution of a CQ Q, a function returning a set of tuples $T \subseteq \{tuples(w_1) \times \ldots \times tuples(w_n)\}$ over $\overline{y}$, where $w_1, \ldots, w_n \in wrap(Q)$. We also use $Q_1 \sqsubseteq Q_2$ to denote a CQ Q_1 is *contained* in another CQ Q_2. Additionally, Q_1 is *maximally-contained* in Q_2 if there does not exist any Q_3 such that $Q_1 \sqsubseteq Q_3 \sqsubseteq Q_2$. We refer the reader to [35] for the formal semantics on the evaluation of CQs and query containment.

A union of conjunctive queries (UCQ) is an expression of the form

$$\overline{Q} = Q_1 \cup \ldots \cup Q_n$$

where $Q_1, \ldots, Q_n$ are union-compatible CQs. Two CQs Q_1 and Q_2 are union-compatible if there is a bijective function between their attributes. From now on, we will interpret a set of CQs as a UCQ. We use $exec(\overline{Q})$ to denote the set of tuples resulting from evaluating the UCQ (i.e., $exec(Q_1) \cup \ldots \cup exec(Q_n)$). Finally, recall that, given a UCQ $\overline{Q}$ and a CQ Q', then $Q' \sqsubseteq \overline{Q}$ if and only if there is an $1 \leq i \leq n$ such that $Q' \sqsubseteq \overline{Q}_i$.

3.2 Integration graph

An integration graph $\mathcal{I}$ is a 4-tuple of edge-labeled directed graphs $\langle \mathcal{G}, \mathcal{S}, \mathcal{M}, \mathcal{E} \rangle$, whose components we describe next. Hereinafter, we assume all operations are applied over a fixed instance of $\mathcal{I}$.

Global graph. The *global graph* $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$ is an unweighted, directed edge-labeled graph where self loops are allowed. $V_\mathcal{G}$ is partitioned into two disjoint sets C (concepts) and F (features). The set F itself is further partitioned into four disjoint subsets, distinguishing derived/base features, and ID/non-ID features. Hence, F_d^{id} and F_b^{id} refer, respectively, to the sets of ID derived and base ID features, while F_d and F_b refer, respectively, to the sets of non-ID derived and base features. Next, labels in $E_\mathcal{G}$ contain the domain $\mathcal{L}$ of the user (i.e., any business concept) as well as the set of *semantic annotations* $\mathcal{A}$. Semantic annotations are system specific labels, for instance to drive the query rewriting process. Note that $\mathcal{A}$ and $\mathcal{L}$ must be disjoint. For now, we focus on the semantic annotation `hasFeature`, relating concepts and their features. Hence, we formalize the edge set $E_\mathcal{G}$ as the union of *(a)* $C \times \mathcal{L} \times C$, assigning labels in $\mathcal{L}$ between concepts; and *(b)* $C \times \{\texttt{hasFeature}\} \times F$, linking concepts and their features. We restrict features to be linked to at most one concept. Hence, given a feature f, we use $conc(f)$ to refer to its associated concept. Conversely, for a given concept c, we use $feat(c)$ to refer to c's set of features. Regarding IDs, in the spirit of composite primary keys, we allow concepts to have more than one ID feature. Moreover, for each concept c, we assume functional dependencies from IDs to non-IDs (i.e., $\{F_d^{id} \cup F_b^{id}\} \rightarrow \{F_d \cup F_b\}$).

Global graph instances. Let V be a countably infinite set of node IDs, v an element in V, and D an infinite set of values. An instance of a concept c with features $f_1, \ldots, f_n$ is a graph $G_c = (V_c, E_c)$, where $V_c = \{v, d_1, d_n\}$, with $d_i \in D$, and $E_c = \{\langle v, f_i, d_i \rangle | i = 1..n\}$, where $f_i \in feat(c)$. Intuitively, this is a graph with exactly one node ID that is connected to values using edges labeled with feature names. Then, a global graph instance G_I from $\mathcal{G}$ with n concepts, is a graph $G_{c_1} \cup \ldots \cup G_{c_n}$, such that, for each pair of concepts c_i, c_j in $\mathcal{G}$ connected with label ℓ, it satisfies that v_i and v_j are connected in $\mathcal{G}_I$ with label ℓ. We denote $\mathbb{G}_I$ the set of all possible global graph instances.

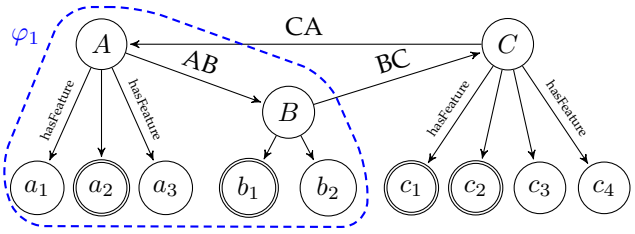


Fig. 2: Global graph and a query. Doubly circled features denote IDs. For the sake of clarity, some `hasFeature` edges have been omitted.

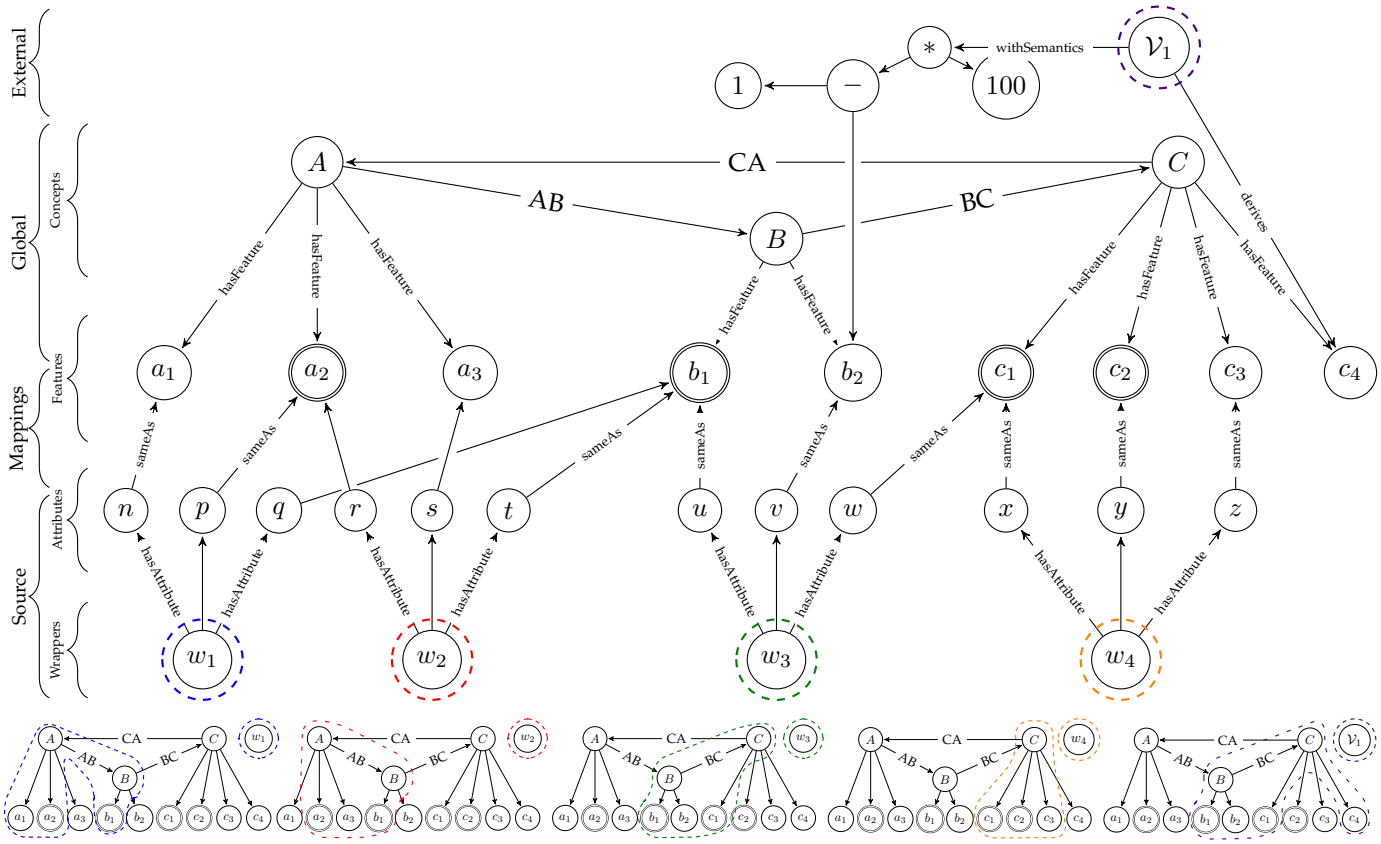


Fig. 3: An exemple integration graph. Doubly circled features denote IDs. The bottom colored graphs represent mappings (i.e., subgraphs of $\mathcal{G}$) for each wrapper dashed with the same color.

Global queries. A *global query* φ is a connected subgraph of $\mathcal{G}$ disregarding edge directions. We say a graph $G_1 = (V_1, E_1)$ is a subgraph of $G_2 = (V_2, E_2)$ (i.e., $G_1 \subseteq G_2$) if $V(G_1) \subseteq V(G_2)$ and $E(G_1) \subseteq E(G_2)$. We use $feat(\varphi)$ to denote the set of features contained in φ. The semantics of a global query are based on graph homomorphisms, the customary for graph databases [36]. Precisely, given two directed edge-labeled graphs $G = (V, E)$ and $G' = (V', E')$, a homomorphism from G to G' is a function $f : V \to V'$ such that for each pair $u, v \in V$, $(u, v) \in E$ implies $(f(u), f(v)) \in E'$. Thus, we will consider as instances of φ the set of graph instances $\varphi_I \subseteq \mathbb{G}_I$ such that, for each $G_I \in \varphi_I$, there exists a homomorphism from φ to G_I.

Example 1. *Figure 2, depicts a global graph with three concepts. The global query φ_1 asks for all features from A and B.*

Source graph. The definition of the source graph $\mathcal{S}$ is analogous to that of $\mathcal{G}$. Intuitively, the source graph encodes in a graph manner wrappers and their schema. Here, however, the vertex set $V_\mathcal{S}$ is composed of $(W \cup A)$, respectively the set of wrappers and attributes (recall that $\mathcal{S}$ is a graph-based representation of the wrappers and their attributes). Here, we introduce the semantic annotation `hasAttribute`, meant to connect a wrapper with its attributes. Thus, in $\mathcal{S}$ the edge set $E_\mathcal{S}$ is composed of $(W \times \{\texttt{hasAttribute}\} \times A)$.

Mappings graph. The mappings graph $\mathcal{M}$ encodes LAV schema mappings between $\mathcal{S}$ and $\mathcal{G}$. Precisely, for a wrapper w, a LAV schema mapping is a pair $\mathcal{M}(w) = \langle \varphi, \mathcal{F} \rangle$, where φ is a global query; and $\mathcal{F}$ is an injective function $\mathcal{F} : att(w) \to feat(\varphi)$. Intuitively, the mappings graph represents in the form of a graph schema mappings between the source and global graphs. Then, we define the functions $glob(\mathcal{M}(w))$ and $map(\mathcal{M}(w))$ respectively denoting, for $\mathcal{M}(w)$, the global query φ and the mapping from attributes to features $\mathcal{F}$. Recall we encode mappings in a graph form, precisely $\mathcal{M}$ contains φ and $\mathcal{F}$. Thus, we represent φ via a subgraph of $\mathcal{G}$, which intuitively identifies the concepts in $\mathcal{G}$ covered by w. To represent $\mathcal{F}$, we extend the set of semantic annotations $\mathcal{A}$ with the `sameAs` label, linking attributes in $\mathcal{S}$ to features in $\mathcal{G}$. Finally, we consider the inverse correspondence $\mathcal{F}^{-1} : f \to A$, in order to identify the set of attributes A that map to a specific feature f.

Definition 1 (COVERING$_\mathcal{I}(W, \varphi)$)**.** *A set of wrappers W covers a global query φ if it is a subset of the union of LAV mappings. Formally, $\forall w \in W : \bigcup glob(\mathcal{M}(w)) \supseteq \varphi$.*

External graph. The external graph $\mathcal{E}$ encodes views together with the semantics of the expressions to compute derived features (i.e., operational expression trees). Formally, a view $\mathcal{V}$ is a triple $\langle f, \varphi, \mathcal{T} \rangle$, where f is a derived feature in the set $\{F_d^{id} \cup F_d\} \in V_\mathcal{G}$, φ is a global query, and $\mathcal{T}$ is an operational expression tree over $feat(\varphi)$. We use $feat(\mathcal{V})$, $glob(\mathcal{V})$ and $exp(\mathcal{V})$ to, respectively, denote f, φ and $\mathcal{T}$. We also use $der(f)$ to denote the inverse function of $feat(\mathcal{V})$. An operational expression tree (or just tree) is a function $\mathcal{T} : T \to T'$, where T and T' are sets of tuples. We

denote as $sem(\mathcal{T})$ the semantics of $\mathcal{T}$ (i.e., the expression it encodes), which are represented via an alphanumerically ordered binary search tree ordered on the expression elements. Expression elements consist of algebraic operators, function calls, variables and constants. We use $\mathcal{T}(T)$ to denote the evaluation of $sem(\mathcal{T})$ on the set of tuples T. A view $\mathcal{V}$ is represented in $\mathcal{E}$ as a node V, where: $feat(\mathcal{V})$ is represented via the semantic annotation `derives`; $glob(\mathcal{V})$ is represented via a subgraph of $\mathcal{G}$ identified by $\mathcal{V}$; and $exp(\mathcal{V})$ is represented via the `withSemantics` annotation.

Example 2. *Figure 3, depicts a complete integration graph.*

3.3 Querying the wrappers via the integration graph

Source queries. A *source query* ψ is a (potentially disconnected) subgraph of $\mathcal{S}$. We denote $att(\psi)$ and $wrap(\psi)$ respectively the attributes and wrappers in ψ. Source queries are isomorphic to CQs for a given integration graph $\mathcal{I}$, hence we define the isomorphism $h_{\mathcal{I}} : \Psi \rightarrow \mathbb{Q}$ from the set Ψ of source queries to the set $\mathbb{Q}$ of CQs. We denote $h_{\mathcal{I}}^{-1}$ its inverse. Precisely, $h_{\mathcal{I}}(\psi)$ yields a CQ $\langle att(\psi), P, wrap(\psi)\rangle$, where P are pairs of attributes that map to the same feature in $\mathcal{G}$. Henceforth, we will only consider equi-join predicates among ID features, formally defined as $\forall p \in predatt(h_{\mathcal{I}}(\psi))\exists w \in wrap(\psi) : map(\mathcal{M}(w))(p) \in F_{id}$. Containment of source queries is equivalent to CQ containment, hence, for two source queries ψ_1 and ψ_2, we say ψ_1 is contained in ψ_2 ($\psi_1 \sqsubseteq \psi_2$) if and only if $h_{\mathcal{I}}(\psi_1) \sqsubseteq h_{\mathcal{I}}(\psi_2)$.

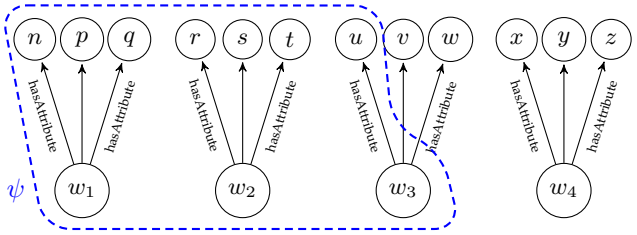
Definition 2 (COVERING$_{\mathcal{I}}(\psi, \varphi)$)**.** *A source query ψ covers a global query φ if* COVERING$_{\mathcal{I}}(wrap(\psi), \varphi)$ *is satisfied.*

Definition 3 (MINIMAL$_{\mathcal{I}}(\psi, \varphi)$)**.** *A source query ψ is minimal w.r.t. a global query φ if removing any wrapper from $wrap(\psi)$ yields a non-covering set of wrappers. Formally, $\nexists w \in wrap(\psi)$:* COVERING$_{\mathcal{I}}(wrap(\psi) \setminus w, \varphi)$.

Semantics of source queries. The semantics of source queries is analogous to that of CQs over the wrappers. Yet, evaluating ψ one would expect to obtain a graph structure instead of a set of tuples. Hence, we consider a homomorphism $g_{\mathcal{I},\varphi} : \mathbb{T}_{\varphi} \rightarrow \mathbb{G}_I$ from the set of tuples $\mathbb{T}_{\varphi}$ resulting from $exec(h_{\mathcal{I}}(\psi))$, to the set of global graph instances $\mathbb{G}_I$. Let T be a set of tuples such that $T \subseteq \mathbb{T}_{\varphi}$, then for each $t \in T$, $g_{\mathcal{I},\varphi}(t)$ yields a graph G such that: *(a)* for each pair of connected concepts $c_i, c_j \in \varphi$ with a label e, G connects the unique node IDs v_i, v_j with the label e (i.e., for each tuple and concept, a new unique node ID is generated); and *(b)* for each concept c linked to a feature f, G connects the node ID c_i to f's corresponding value in t using the label f.

Definition 4 (CONTAINMENT$_{\mathcal{I}}(\psi, \varphi)$)**.** *A source query ψ is contained into a global query φ (i.e., $\psi \sqsubseteq \varphi$) if all tuples resulting from the execution of $h_{\mathcal{I}}(\psi)$ are instances of φ. Formally, $\forall t \in exec(h_{\mathcal{I}}(\psi)) : g_{\mathcal{I},\varphi}(t) \in \varphi_I$. ψ is maximally-contained in φ if there does not exist a source query $\psi' \neq \psi$ such that $\psi \sqsubseteq \psi' \sqsubseteq \varphi$.*

Example 3. *Figure 4 depicts a contained source query ψ of φ_1 (from Figure 2). Applying the isomorphism $h_{\mathcal{I}}$, after removing redundant attributes, we obtain the same query in its CQ form. Below, we present exemplary tuples from $exec(h_{\mathcal{I}}(\psi))$, followed by the global graph instance form of t_1 (i.e., $g_{\mathcal{I},\varphi}(t_1)$).*



$$h_{\mathcal{I}}(\psi) = \pi_{n,p,s,u,v}((W_1 \times W_2 \times W_3)|p = r \wedge q = u \wedge t = u)$$

$exec(h_{\mathcal{I}}(\psi))$:

	n	p	s	u	v
t_1	n_1	p_1	s_1	u_1	v_1
t_2	n_2	p_2	s_2	u_2	v_2

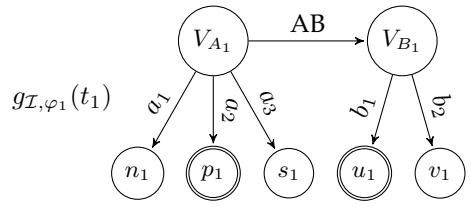


Fig. 4: Contained source query, execution and global instance of the global query from Figure 2

Rewriting algorithm. A rewriting algorithm is a function $\mathcal{R}_{\mathcal{I}} : \phi \rightarrow \overline{\Psi}$ from the set ϕ of all global queries to the set $\overline{\Psi}$ of sets of source queries, such that $\forall \varphi \in \phi$, $\mathcal{R}_{\mathcal{I}}(\varphi)$ consists only of rewritings of φ. We define the notions of *minimally-sound* and *minimally-complete* rewriting algorithms.

Definition 5 (MINIMALLY-SOUND$(\mathcal{R}_{\mathcal{I}})$)**.** *A rewriting algorithm $\mathcal{R}_{\mathcal{I}}$ is minimally-sound if $\forall \varphi \in \phi$ and $\forall \psi \in \mathcal{R}_{\mathcal{I}}(\varphi)$, then* MINIMAL$_{\mathcal{I}}(\psi, \varphi)$ *is satisfied.*

Definition 6 (MINIMALLY-COMPLETE$(\mathcal{R}_{\mathcal{I}})$)**.** *A rewriting algorithm $\mathcal{R}_{\mathcal{I}}$ is minimally-complete if $\forall \varphi \in \phi$ and every ψ such that it is a rewriting of φ and* MINIMAL$_{\mathcal{I}}(\psi, \varphi)$ *is satisfied, it holds that $\psi \in \mathcal{R}_{\mathcal{I}}(\varphi)$.*

Theorem 1. *Let φ be a global query, and let $\mathcal{R}_{\mathcal{I}}$ be a rewriting algorithm. Each source query ψ in the set $\Psi = \mathcal{R}_{\mathcal{I}}(\varphi)$ is maximally-contained in φ (i.e., $\psi \sqsubseteq \varphi$), if and only if $\mathcal{R}_{\mathcal{I}}$ is minimally-sound and minimally-complete.*

Proof. Let $\Psi = \{\psi_1, \ldots, \psi_n\}$ be the set of source queries resulting from $\mathcal{R}_{\mathcal{I}}(\varphi)$. We first show that each $\psi_i \sqsubseteq \varphi$ showing that each tuple $t \in exec(h_{\mathcal{I}}(\psi_i))$ is contained in the set of instances of φ. In other words, let G be the graph generated by applying the homomorphism $g_{\mathcal{I},\varphi}(t)$, then we must show that G is an instance of φ, which reduces to show there exists a homomorphism from φ to G. The if can be shown relying on the fact that $\mathcal{R}_{\mathcal{I}}$ is minimally-sound (i.e., t is the result of a covering source query). In this case, the number of edges in φ will be less or equal than the number of edges in G, guaranteeing the existence of the homomorphism. This is not the case under the assumption that $\mathcal{R}_{\mathcal{I}}$ is not minimally-sound. We additionally reason that each $\psi_i \sqsubseteq \varphi$ is maximally-contained in φ, using the assumption that $\mathcal{R}_{\mathcal{I}}$ is minimally-complete. This guarantees there does not exist a source query $\psi' \notin \Psi$ such that COVERING$_{\mathcal{I}}(\psi', \varphi)$. □

Problem statement. The problem of rewriting queries φ over an integration graph $\mathcal{I}$ reduces to finding a minimally-sound and minimally-complete rewriting algorithm $\mathcal{R}_{\mathcal{I}}$.

4 REWRITING CONJUNCTIVE QUERIES

In this section, we present REWRITECQ, a three-phase minimally-sound and minimally-complete rewriting algorithm. We detail each of its phases, and later present a discussion on its computational complexity and properties.

Algorithm 1 depicts the main method that rewrites global queries φ. It starts unfolding the GAV mappings in $\mathcal{E}$, and encoding the computation of derived features as virtual wrappers. Then, LAV mappings are used to find equi-join conditions among wrappers to yield a contained UCQ $\overline{Q}$. REWRITECQ is inspired by the *bucket algorithm* for LAV mediation [13], which finds rewritings for each subgoal in the query, and stores them in buckets. Then, it finds a set of conjunctive queries such that each of them contains one conjunct from every bucket. In our case, concepts are analogous to buckets, however equi-join conditions must be automatically discovered. Hence, we will first separately find rewritings that cover the requested concepts in φ to later find all possible minimal combinations among them.

Algorithm 1 REWRITECQ

Pre: $\mathcal{I}$ is an integration graph, φ is a global query
Post: Ψ is a set of source queries
1: **function** REWRITECQ($\mathcal{I}, \varphi$)
2: $\mathcal{I}' \leftarrow$ UNFOLD($\mathcal{I}, \varphi$)
3: $G \leftarrow$ GENERATEREWRITINGS($\mathcal{I}', \varphi$)
4: $\Psi \leftarrow$ COMBINEREWRITINGS(G)
5: **return** Ψ

Throughout this section, we will exemplify our approach using the global query depicted in Figure 5.

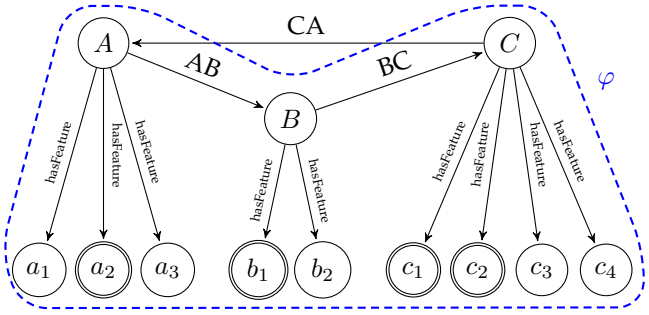


Fig. 5: Global query φ used as running example

4.1 Unfolding derived features

The first phase (see Algorithm 2) of the rewriting algorithm consists on computing the derived features contained in φ. The unfolding process consists on the generation of an integration graph $\mathcal{I}'$ where there are no views associated to derived features in φ. To this end, the first part of the algorithm consists on, for each view $\mathcal{V}$ (line 4) associated to every derived feature f in φ, obtaining a UCQ $\overline{Q}$ (line 6) from the global query $glob(\mathcal{V})$. Executing such UCQ (line 7) yields a set of tuples T, which will be used in the computation of the derived feature via the operational expression tree $exp(\mathcal{V})$.

The second part is devoted to generate a virtual wrapper w_v (line 8) with the result of computing the derived feature over $\overline{Q}$. This is achieved taking a query in $\overline{Q}$ (line 9) to extract its schema (recall that queries in $\overline{Q}$ are union-compatible, hence all have the same schema). Then, the loop in line 10 deals with the population of $att(w_v)$ with fresh (i.e., new) attribute names avoiding repeating equivalent attributes (line 12). The loop also populates the mapping. Next, we also add a new attribute for the derived feature (line 17) and its mapping. The set of tuples returned by w_v is defined as those from $\overline{Q}$ including an extra computation for the derived feature. Finally, with some abuse of notation in order to extend the isomorphism $h_{\mathcal{I}}$ to wrappers, w_v is added as a new wrapper to $\mathcal{I}'$, while $\mathcal{V}$ is removed.

Algorithm 2 Unfold derived features

Pre: $\mathcal{I}$ is an integration graph, φ is a global query
Post: $\mathcal{I}'$ is an integration graph where all features covered by φ in $\mathcal{I}$ have been unfolded
1: **function** UNFOLD($\mathcal{I}, \varphi$)
2: $\mathcal{I}' \leftarrow \mathcal{I}$
3: **for** $f \in feat(\varphi) | f \in \{F_d^{id} \cup F_d\}$ **do**
4: $\mathcal{V} \leftarrow der(f)$
5: $\varphi_d \leftarrow glob(\mathcal{V})$, $\mathcal{T} \leftarrow exp(\mathcal{V})$
6: $\overline{Q} \leftarrow$ REWRITECQ($\mathcal{I}', \varphi_d$)
7: $T \leftarrow exec(\overline{Q})$
8: $w_v \leftarrow$ `new Wrapper()`
9: $Q \leftarrow$ `takeAny(`$\overline{Q}$`)`
10: **for** $w \in wrap(Q)$ **do**
11: **for** $a \in att(Q) | a \in att(w)$ **do**
12: **if** $\nexists a' \in att(w_v) | map(\mathcal{M}(w))(a) = map(\mathcal{M}(w_v))(a')$
13: $a_n \leftarrow$ `freshAttributeName()`
14: $att(w_v) \cup\!= a_n$
15: $map(\mathcal{M}(w_v)) \cup\!= \langle a_n \rightarrow map(\mathcal{M}(w))(a) \rangle$
16: $a_d \leftarrow$ `freshAttributeName()`
17: $att(w_v) \cup\!= a_d$
18: $map(\mathcal{M}(w_v)) \cup\!= \langle a_d \rightarrow f \rangle$
19: $glob((\mathcal{M}(w_v)) \leftarrow \varphi_d \cup \langle conc(f), \texttt{hasFeature}, f \rangle$
20: $exec(w_v) \leftarrow T \cup \mathcal{T}(T)$
21: $\mathcal{I}' \cup\!= h_{\mathcal{I}}^{-1}(w)$
22: $\mathcal{I}' \setminus\!= \mathcal{V}$
23: **return** $\mathcal{I}'$

Example 4. *Recall that feature c_4 in the running example depicted in Figure 5 is a derived feature. After unfolding its corresponding view, the integration graph $\mathcal{I}'$ would contain a wrapper w_v, where $att(w_v) = \{i, j, k, l\}$ resulting from the computation of the query $\pi_{u,v,w,\mathcal{T}}((W_3 \times W_4) | w = x)$.*

4.2 Generating rewritings

This second phase (see Algorithm 3) receives as input an integration graph $\mathcal{I}$ and a global query φ, where all features covered by φ in $\mathcal{I}$ have been unfolded. Here, the objective is to generate sets of rewritings for each concept in $conc(\varphi)$. To this end, we define the *rewritings graph* $G_\psi = (V_\psi, E_\psi)$, an auxiliary graph data structure such that vertices V_ψ are sets of source queries Ψ. Intuitively, the rewritings graph will encode, for each concept c in $conc(\varphi)$, all rewritings that are covering and minimal with respect to c and its queried features. The output of Algorithm 3 is a graph G_ψ.

For each concept c covered by φ, Algorithm 3 populates the set of attributes A such that have mapping to some queried feature (lines 4-6). Then, it searches for candidate source queries (line 7). This is, CQs containing all attributes

that map to some queried feature in $feat(c)$ for a wrapper (lines 8-12). The last step, consists of systematically processing the set of candidate source queries in order to generate rewritings (i.e., covering source queries that only use IDs on equi-joins) (lines 14-17). To this end, the method COVERINGCQS is leveraged. Finally, the rewritings graph G_ψ is constructed preserving the original edge labels in φ. This is achieved, for each edge e not labeled `hasFeature`, obtaining its source node (i.e., FROM(e)) and its target node (i.e., TO(e)).

Algorithm 3 Generate rewritings

Pre: $\mathcal{I}$ is an integration graph with no derived features, φ is a global query
Post: G_ψ is a rewritings graph

1: **function** GENERATEREWRITINGS($\mathcal{I}, \varphi$)
2: $G_\psi \leftarrow$ empty rewritings graph
3: **for** $c \in conc(\varphi)$ **do**
4: $A \leftarrow \emptyset$
5: **for** $f \in feat(\varphi) | f \in feat(c)$ **do**
6: $A \cup= \mathcal{F}^{-1}(f)$
7: $\Psi_{candidates} \leftarrow \emptyset$
8: **for** $a_i \in A$ **do**
9: $Q \leftarrow \langle \{a_i\}, \emptyset, \{wrap(a_i)\} \rangle$
10: **for** $a_j \in A | a_i \neq a_j \wedge wrap(a_i) = wrap(a_j)$ **do**
11: $att(Q) \cup= a_j$
12: $\Psi_{candidates} \cup= h_{\mathcal{I}}^{-1}(Q)$
13: $\Psi_{covering} \leftarrow \emptyset$
14: **while** $\Psi_{candidates} \neq \emptyset$ **do**
15: $\psi \leftarrow$ `takeAny` ($\Psi_{candidates}$)
16: $I \leftarrow (\{c\} \times \{\texttt{hasFeature}\} \times feat(\varphi))$
17: $\Psi_{covering} \cup=$ COMPOSE($I, \psi, \psi_{candidates}$)
18: $V(G_\psi) \cup= \Psi_{covering}$
19: **for** $e \in E(\varphi) | label(e) \neq \texttt{hasFeature}$ **do**
20: $E(G_\psi) \cup= \langle$FROM(e), e, TO(e)$\rangle$
21: **return** G

Composing source queries into rewritings. The process of generating rewritings (see Algorithm 4) is a recursive task that given an input source query ψ and a set of candidate source queries, incrementally generates covering combinations (i.e., rewritings). Ultimately, each of this generated combinations must cover the graph I. Here, I represents the graph induced by the concept c and its queried features. It is important to note that this method finishes when composing a new rewriting would not yield new features, which ensures *minimality*. Generating the combination of two source queries might entail discovering join conditions among them (see method FINDJOINS in Algorithm 5).

Join discovery. Given two source queries ψ_a and ψ_b, the method FINDJOINS (see Algorithm 5) performs the process of finding equi-join predicates among them. This is, it finds all shared IDs covered by $wrap(\psi_a)$ and $wrap(\psi_b)$. First, the algorithm identifies the sets of IDs F_a^{id} and F_b^{id} that both source queries contribute to (line 3-6). Then, for each shared ID f, it finds pairs of attributes that have a mapping to it, which will define a new equi-join predicate.

Algorithm 4 Compose source queries

Pre: I is the graph to check coverage, ψ is a source query, $\psi_{candidate}$ is a set of candidate source queries
Post: $\psi_{candidate}$ is empty, $\psi_{covering}$ contains all potential combinations of covering rewritings with respect to I

1: **function** COMPOSE(I, ψ, $\psi_{candidate}$)
2: $\psi_{covering} \leftarrow \emptyset$
3: **if** COVERING(ψ, I) ▷ From Definition 2
4: $\psi_{covering} \cup= \psi$
5: **else if** $\psi_{candidate} \neq \emptyset$
6: **for** $\psi' \in \psi_{candidate}$ **do**
7: **if** $\psi \cup \psi'$ provides more features than ψ
8: $\psi_{new} \leftarrow$ FINDJOINS(ψ, ψ')
9: COMPOSE($I, \psi_{new}, \psi_{candidate} \setminus \psi'$)
10: **return** $\psi_{covering}$

Algorithm 5 Find joins

Pre: ψ_a and ψ_b are source queries
Post: ψ is a rewriting from the composition $h(\psi_a) \oplus h(\psi_b)$ with the necessary equi-joins among them

1: **function** FINDJOINS(ψ_a, ψ_b)
2: $F_a^{id} \leftarrow \emptyset$
3: **for** $w \in wrap(\psi_a)$ **do**
4: **for** $a \in att(w)$ **do**
5: **if** $map(\mathcal{M}(w))(a) \in \{F_d^{id} \cup F_d\}$
6: $F_a^{id} \cup= map(\mathcal{M}(w))(a)$
7: $F_b^{id} \leftarrow$ repeat lines 3-6 using ψ_b
8: $\bowtie \leftarrow \emptyset$
9: **for** $f \in \{F_a^{id} \cap F_b^{id}\}$ **do**
10: **for** $w_a \in wrap(\psi_a)$ **do**
11: **for** $a_a \in att(w_a) | a_a \in \mathcal{F}^{-1}(f)$ **do**
12: **for** $w_b \in wrap(\psi_b)$ **do**
13: **for** $a_b \in att(w_b) | a_b \in \mathcal{F}^{-1}(f)$ **do**
14: **if** $map(\mathcal{M}(w))(a_a) = map(\mathcal{M}(w))(a_b)$
15: $\bowtie \cup= \langle a_a, a_b \rangle$
16: $Q \leftarrow h(\psi_a) \oplus h(\psi_b)$
17: $pred(Q) \cup= \bowtie$
18: **return** $h^{-1}(Q)$

Example 5. *On the running example's global query, the output of Algorithm 3 would be the graph G_ψ depicted in Figure 6.*

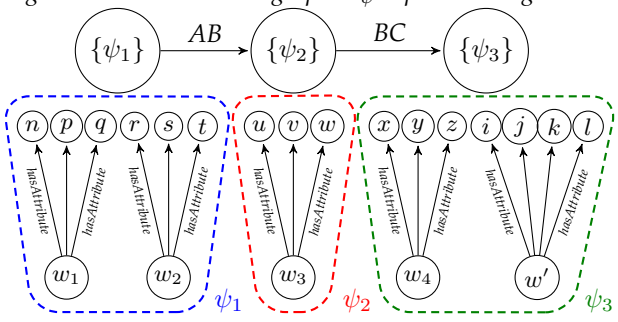


Fig. 6: Rewritings graph for the global query in Figure 5

4.3 Combining rewritings

Algorithm 6 combines rewritings covering connected concepts. It receives as input a rewritings graph G_ψ, and

systematically compacts edges to generate new sets of minimal source queries. At each iteration, a synthetic node is generated from compacting the sets of rewritings Ψ_a and Ψ_b, respectively the FROM and TO nodes of an edge e. The algorithm ends when the graph has no edges. Precisely, the set of wrappers W_e identifies wrappers that cover the edge e. Hence, only combinations containing some wrapper covering e will be considered, which reduces the search space. Note CHOOSEEDGE might range from a purely random selection to an informed heuristic prioritizing early pruning.

Algorithm 6 Combine rewritings

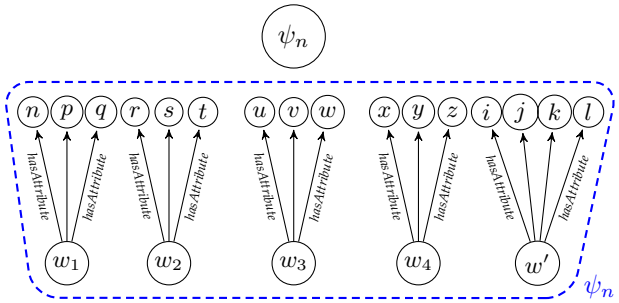
Pre: G_ψ is a rewritings graph
Post: $V(G_\psi)$ is a set of source queries
1: **function** COMBINEREWRITINGS(G_ψ)
2: **while** $E(G) \neq \emptyset$ **do**
3: $e \leftarrow$ CHOOSEEDGE(G_ψ)
4: $W_e \leftarrow \emptyset$
5: **for** $w \in wrap(\text{FROM}(e))$ **do**
6: **if** $e \in glob(\mathcal{M}(w))$
7: $W_e \cup= w$
8: repeat lines 5-7 using TO(e)
9: $I \leftarrow$ is the subgraph of φ that FROM(e) and TO(e) cover, connected via the edge e
10: $\Psi \leftarrow$ COMBINE(FROM(e), TO(e), W_e, I)
11: Remove FROM(e), TO(e) from G_ψ, add a new vertex Ψ preserving connectivity.
12: **return** $V(G_\psi)$

Combining sets of rewritings. Given two sets of source queries Ψ_a and Ψ_b, COMBINE (see Algorithm 7) generates their minimal combinations. Precisely, only pairs in the cartesian product $\Psi_a \times \Psi_b$ covering the edge and minimal will be considered. Coverage is based on checking if any wrapper on both ends covers it. Minimality is checked on a graph I, which denotes the subgraph of the original global query φ that the synthetic node represents. For instance, at the second iteration, after compacting nodes B and C to generate a new node BC, minimality will still be checked on the original subgraph $B \rightarrow_{BC} C$ (including their queried features). To generate such combination, the previously described method FINDJOINS is used (see Algorithm 5).

Algorithm 7 Combine sets of rewritings

Pre: Ψ_a and Ψ_b are sets of source queries, W is a set of wrappers that cover the edge connecting Ψ_a and Ψ_b in the rewritings graph, I is the graph to check minimality
Post: Ψ is a set with all valid combinations of Ψ_a and Ψ_b
1: **function** COMBINE(Ψ_a, Ψ_b, W, I)
2: $\Psi \leftarrow \emptyset$
3: **for** $\langle \psi_a, \psi_b \rangle \in \Psi_a \times \Psi_b$ **do**
4: **if** $wrap(\psi_a) \subseteq W \vee wrap(\psi_b) \subseteq W$
5: **if** MINIMAL($\psi_a \cup \psi_b, I$) ▷ From Definition 3
6: $\Psi \cup=$ FINDJOINS(ψ_a, ψ_b)
7: **return** Ψ

Example 6. *In the running example, Algorithm 6 will perform two iterations (i.e., edges AB and BC). The resulting rewriting is depicted in Figure 7.*



$$h_{\mathcal{I}}(\psi_n) = \{\pi_{n,p,s,u,v,x,y,z,l}((w_1 \times w_2 \times w_3 \times w_4 \times w') | \\ p = r \wedge q = u \wedge t = u \wedge w = x \wedge x = i)\}$$

Fig. 7: G_ψ generated in Algorithm 6, and isomorphic UCQ

5 ALGORITHM ANALYSIS

In this section, we discuss the computational complexity of REWRITECQ and show how minimally-soundness and minimally-completeness are satisfied.

5.1 Computational complexity

To classify REWRITECQ to its complexity class let us first define a function $dep(w)$ that returns a set of wrappers $\overline{w} \subseteq W$ for which w depends on (i.e., they are used as part of the computation of $exec(w)$). Likewise, we also define the closure dependency operation $dep^*(w)$ as the recursion

$$dep^*(w) = \begin{cases} \emptyset, & \text{if } dep(w) = \emptyset, \\ dep^*(w') \cup w', \text{for } w' \in dep(w) & \text{otherwise.} \end{cases}$$

Then, we say that a wrapper w has a *cyclic dependency* if $w \in dep^*(w)$. Particularly, our unfolding algorithm performs a particular instance of the chase [37]. Hence, for certain integration graphs where there exist cyclic dependencies there may not exist a finite chase and the algorithm might fall into infinite recursion.

Theorem 2. *Rewriting a global query is NP-hard in query complexity if all wrappers covering φ have no cyclic dependencies.*

Proof. Theorem 2 is proved by reduction from *Set Cover* [38], a well-known NP-hard problem defined as: given a set S of n points and $\mathcal{F} = \{S_1, S_2, \ldots, S_m\}$ a collection of subsets of S, select as few as possible subsets from $\mathcal{F}$ such that every point in S is contained in at least one of the subsets. The reduction works as follows. Let us consider a global query φ, where for each point in S we generate a triple $p_i = \langle s, \ell, t \rangle \in \varphi$ (note graph edges can be disregarded and checked at the end). Then, from the set $\{S_1, \ldots, S_m\}$ we consider the set of all wrappers covering some point in S. We can see that finding combinations of subsets is equivalent to finding combinations of wrappers such that the complete set of attributes in the query is covered. Furthermore, set cover seeks as few as possible subsets, which is equivalent to our definition of minimality. As a matter of fact, we are interested in enumerating all possible solutions of the problem (i.e., minimally-completeness), while in some instances of set cover finding one is enough. □

Next, we aim to get an accurate cost formula for REWRITECQ. Let W be the average number of wrappers

covering each concept (not including uncovered concepts), F be the number of features in a global query φ, and C be the number of concepts covered in φ. Recall that Algorithm 3 generates all covering source queries per concept, by incrementally obtaining all different ways to perform equi-joins among them. Next, Algorithm 6 further finds all combinations of these sets of rewritings. From the previous rationale, we can conclude that the complexity of REWRITECQ is $\mathcal{O}(\binom{W}{F}^C)$. Its worst case corresponds to the scenario where each wrapper only contributes to one queried feature, and thus all possible combinations are explored.

5.2 Minimally-sound and minimally-complete

Here, we show that REWRITECQ is a minimally-sound and minimally-complete rewriting algorithm. Precisely, we show the following invariants: *(a)* $\overline{Q}$ does not contain any non-minimal CQ, and *(b)* $\overline{Q}$ contains all minimal CQs. We assume that the integration graph covered by a query φ has no derived features (i.e., all have been unfolded).

Proof. The trivial case occurs when φ covers a single concept c. Here, only Algorithm 3 will be executed to generate covering queries for c. Then, the set $\Psi_{candidates}$ contains all candidate source queries that cover c and some of its queried features. Next, Algorithm 4 systematically combines source queries. Indeed, this process only generates minimal rewritings (as any combination not contributing with new features is discarded), which guarantees the first invariant. Regarding the second invariant, it is guaranteed by the recursive nature of Algorithm 4, which explores all combinations of candidate source queries to generate rewritings (i.e., with all possible equi-join conditions).

Querying more than one concept involves Algorithm 6. We assume a rewritings graph G_ψ with vertices $\Psi_1, \ldots, \Psi_n$ containing sets of minimal rewritings. Given an edge in G_ψ, we systematically generate all possible combinations of rewritings from the FROM and TO vertices. We show that all minimal rewritings are obtained by reductio ad absurdum. Let us assume the output of Algorithm 6 does not contain a minimal source query ψ. This directly contradicts the fact that combining sets of rewritings in Algorithm 7 checks for minimality (line 5), thus guaranteeing the first invariant. Then, recall that all combinations of rewritings have been computed by a cartesian product (line 3 of Algorithm 7). Hence, ψ has necessarily been generated here, and thus been added to the set Ψ if minimality is satisfied. This contradicts the assumption and guarantees the second invariant. □

6 EXPERIMENTAL EVALUATION

In this section, we measure the performance of REWRITECQ and compare it to alternative approaches for answering queries using views. All details and reproducibility instructions can be found in the companion website[1].

6.1 Experimental setting

To assess our algorithms and facilitate their comparison to alternatives, we generate artificial data via a principled method which is depicted in Algorithm 8. Precisely, we systematically generate synthetic experimental scenarios with different characteristics. Each scenario consists of a global graph, a set of wrappers, mappings, and a global query. To evaluate our approach under different situations, we customize the generation of experimental scenarios using the following variables: 1) number of features per concept ($|F|$); 2) number of edges covered by a query ($|E_Q|$); 3) overall number of wrappers ($|W|$); 4) number of edges covered by a wrapper ($|E_W|$); 5) fraction of features in a concept covered by a query ($Frac_Q$); and 6) fraction of features in a concept covered by a wrapper ($Frac_W$). Then, the process of generating an experimental scenario consists of obtaining random subgraphs of a large enough clique playing the role of $\mathcal{G}$, which guarantees the desired randomness.

Algorithm 8 Generate an experimental scenario

Pre: $\mathcal{G}$ is a clique, $|F|$, $|E_Q|$, $|W|$, $|E_W|$, $Frac_Q$, $Frac_W$
Post: φ is a global query, W is a set of wrappers covering φ
1: **function** GENERATEEXPERIMENTALSCENARIO($\mathcal{G}$, $|F|$, $|E_Q|$, $|W|$, $|E_W|$, $Frac_Q$, $Frac_W$)
2: $\varphi \leftarrow$ connected random subgraph of $\mathcal{G}$ with $|E_Q|$ edges
3: $\varphi' \leftarrow$ with a probability $Frac_Q$ of appearing, expand φ with up to $|F|$ features
4: $W \leftarrow \emptyset$
5: **for** $i \leftarrow 1$ to $|W|$ **do**
6: $w \leftarrow$ connected random subgraph of φ with $|E_W|$ edges
7: $w' \leftarrow$ with a probability $Frac_W$ of appearing, expand φ with up to $|F|$ features
8: $W \cup= w'$
9: **return** $\langle \varphi', W \rangle$

For each combination of experimental variables, we generate an experimental scenario and invoke REWRITECQ. For each run, we measure the size of the resulting UCQs (U) and the processing time (R) in seconds. To account for variability, we generate three experimental scenarios and measure the median of R. Experiments were performed on a GNU/Linux machine with an Intel Core i5 processor running at 3.5 GHz and with 16GB of RAM memory. We implemented a prototype of the rewriting algorithms [39], which is based on SPARQL. There, each construct is represented as an RDF graph.

Alternatives. We compare our approach with the following state-of-the-art solutions for answering queries using views, whose source code is openly available: MiniCon [15] and Graal [22]. The former being a representative of DB-based approaches and the later of the KR-based ones, according to the classification used in Section 2. No fine tuning was performed in such systems, running the code as provided out-of-the-box. To enable a fair comparison of our approach and the alternatives we convert the output of Algorithm 8 into a set of Datalog rules. Recall, however, that our setting does not explicit join variables, hence for each directed edge between a pair of concepts A, B, we materialize the ID features of B in A's variables. Then, for each concept covered in a query or a wrapper, we generate a subgoal with its corresponding attributes. For the case of wrappers, we additionally rename attributes using the mapping from attributes to features (i.e., $map(\mathcal{M}(w))$).

Differences on query rewriting semantics. There exist some circumstances where the query rewriting semantics

1. https://www.essi.upc.edu/dtim/odin/

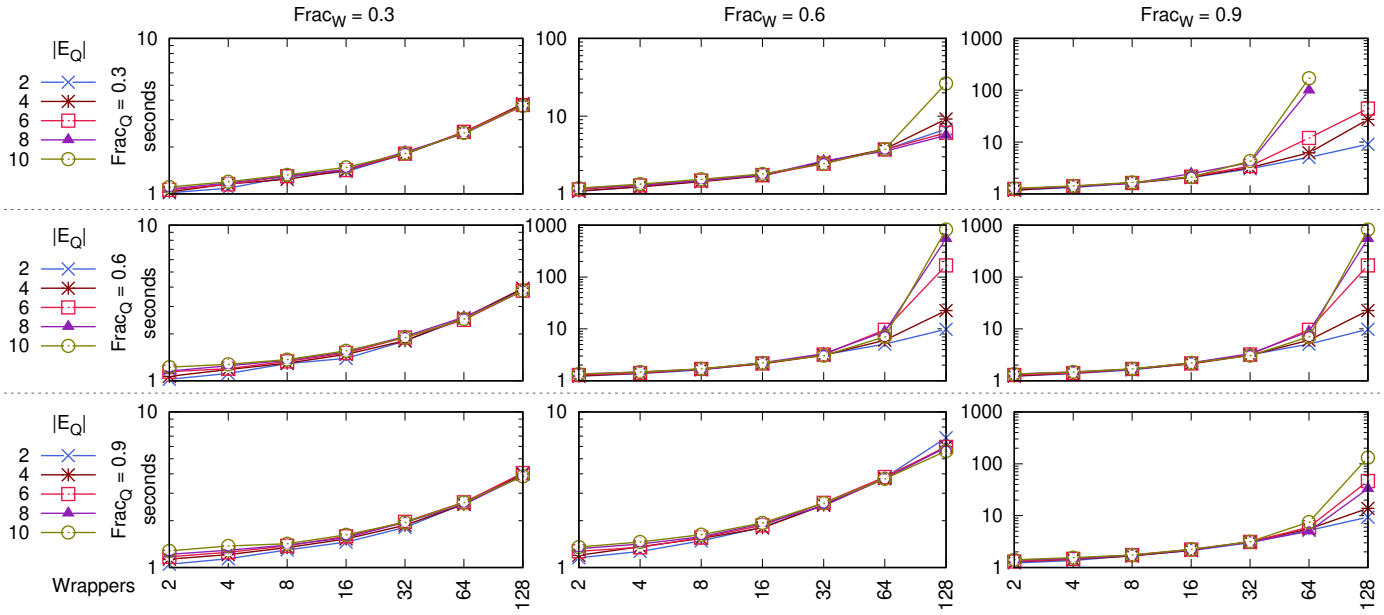


Fig. 8: Evolution of R (y-axis) w.r.t. $|W|$ (x-axis) and $|E_Q|$ (legend), for $|F| = 20$ and $|E_W| = 2$. Missing data points denote that the program ran out of memory due to the size of intermediate results.

of our approach and the alternatives might differ. We next list the three situations we consider using Datalog notation:

- *Equivalent views*, where different views contain the same subgoals and their head variables fully overlap (i.e., they are the same).
- *Intra-relation composition*, where different views contain the same subgoals but the head variables partially overlap. All subgoals in the views correspond to the same relation.
- *Inter-relation composition*, where different views contain the same subgoals but the head variables partially overlap. Additionally, there is at least one view projecting attributes from two different relations.

Table 2, depicts, for such different scenarios the behavior of the different approaches using an exemplary set of views. Precisely, when managing equivalent views, for the exemplary set of views all approaches would return the union of w_1 and w_2. However, when dealing with subsets of attributes, either all from the same subgoal or from different subgoals, both MiniCon and Graal do not consider their composition joining via the shared attributes. Oppositely, in both scenarios we would join w_1 and w_2 using A under the assumption that A is tagged as an ID feature. Precisely, the process of composing source queries deals with the intra-relation composition scenario (see Algorithm 4), while the combination of rewritings deals with the inter-relation composition (see Algorithm 6). Note that we, additionally, have validated that the solutions generated by the alternatives are always contained in ours.

6.2 Experimental results

The results showed a high correlation between the size of the resulting UCQs U and the processing time R (i.e., a Pearson correlation coefficient of $\rho = 0.997$), thus, for space reasons, we only report on R.

Kind of semantics	Example views	MiniCon	Graal	RewriteCQ
Equivalent views	$w_1(A,B) : r(A,B)$ $w_2(A,B) : r(A,B)$	Union		
Intra-relation composition	$w_1(A,B) : r(A,B,C)$ $w_2(A,C) : r(A,B,C)$	$\emptyset$		Join
Inter-relation composition	$w_1(A,B) : r(A,B),$ $s(B,C)$ $w_2(A,C) : r(A,B),$ $s(B,C)$	$\emptyset$		Join

TABLE 2: Comparison of approaches based on different kinds of query rewriting semantics. The second column, depicts an exemplary minimal set of Datalog rules scenarios.

Evolution of response time based on wrappers. We first analyse how R evolves based on the number of wrappers. To this end, we plot, in a logarithmic scale, its evolution for different values of $|W|$. As depicted in Figure 8, there is an exponential trend for R as the number of sources (i.e., wrappers) grows. Nonetheless, we can see our approach can efficiently deal with a large number of sources (i.e., 128) while the number of edges in the query is relatively small. With an increased number of covered edges in φ, the cost also grows exponentially, as occurs on algorithms for answering queries using views. The limitation on number of wrappers is observed as the number of edges covered by the query (i.e., $|E_Q|$) grows. We also observe that, on average, rewriting performance decreases when wrappers cover a large fragment of $\mathcal{G}$ (i.e., $Frac_W = 0.9$). Precisely, as shown in Figure's 8 top-right corner, such worst case is observed when the query covers a small fragment of $\mathcal{G}$ (i.e., $Frac_Q = 0.3$) but the wrappers cover a large fragment (i.e., $Frac_W = 0.9$). As expected, this case might generate many combinations of wrappers composing the same concept to cover all requested features (i.e., the *intra-relation composition*

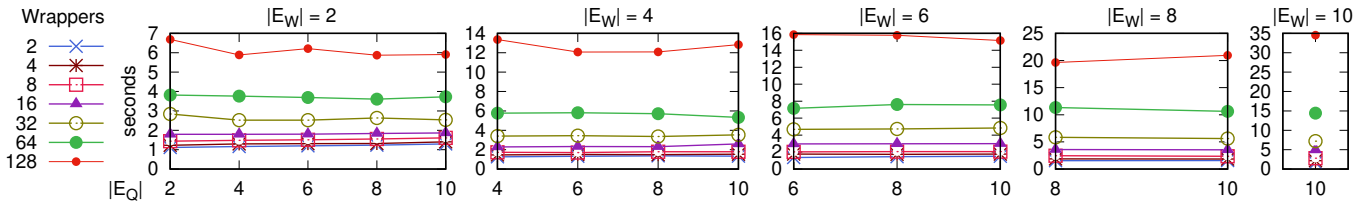


Fig. 9: Evolution of R (y-axis) w.r.t. $|E_Q|$ (x-axis) and $|W|$ (legend), for $|F| = 20$, $Frac_Q = 0.6$ and $Frac_W = 0.6$

semantics). Contrarily, with lower values of $Frac_W$, it is harder to find combinations covering the requested features, and hence the rewriting time significantly decreases.

Evolution of response time based on query size. In this second experiment, we are concerned with studying the impact of the size of the query on the time to perform a rewriting. To this end, we plot the evolution of R for different values of $|E_Q|$ and $|E_W|$. Here, we focus on an intermediate case and hence we fix $Frac_Q$ and $Frac_W$ to 0.6. As depicted in Figure 9, the cost of rewriting is linear in spite of some variability. This shows that the only exponential factor on query rewriting is the number of sources (i.e., $|W|$), a well-known bottleneck in query rewriting algorithms.

Comparison to alternatives on query rewriting. In previous experiments, we observed that both the size of the query and the wrappers (i.e., $|E_Q|$ and $|E_W|$) have no major performance impact (i.e., they are not an exponential factor) for our rewriting process. Hence, here we fix $E_Q = 2$ and $E_W = 2$ to focus on comparing a varying number of wrappers against the alternatives. Furthermore, as presented in Table 2, the only meaningful comparison with the alternatives occurs when no intra or inter-relation composition is required. Precisely, following Algorithm's 8 notation, this corresponds to the case when both $Frac_Q = 1$ and $Frac_W = 1$. We compared the runtime for small values of $|F|$, as our tests showed that the alternatives struggled to manage a large number of features (e.g., $15 - 20$). In order to control the execution time of the alternatives, we set a timeout value equal to $10x$ the time we take to run Algorithm 8 and rewrite the query. We believe this is a large enough value to demonstrate the better performance of our method, while at the same time avoid too lengthy executions of the alternatives. Then, Figure 10, depicts the runtime comparison. Note that, missing data points correspond to execution timeouts. First, we can observe that both alternatives have a much steeper exponential trend than ours. While we efficiently deal with 64 wrappers, MiniCon only manages to successfully execute around half of them. Graal fails to manage more than 10 wrappers. We believe the major performance drawback of such methods is the number of intermediate results they manage (i.e., candidate queries). Indeed, we have observed an exponential number of existential rules in their executions. Under these circumstances, exploration of the search space in a breadth-first search manner, as Graal does, becomes extremely costly. Oppositely, and considering we generate more solutions due to our rewriting semantics, thanks to the ability of querying the mappings, which are stored as graphs, we can select only relevant views in an incremental and more efficient manner.

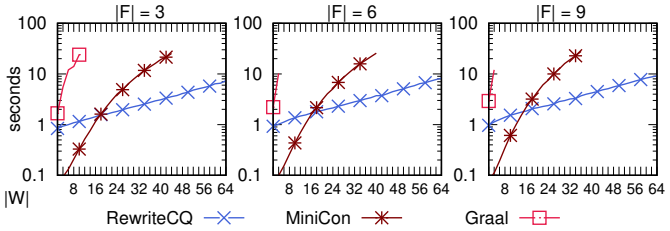


Fig. 10: Runtime evolution (y-axis) w.r.t. $|W|$ (x-axis) and alternative approach under comparison (legend), for $Frac_Q = 1$, $Frac_W = 1$, $E_Q = 2$, and $E_W = 2$. Missing data points denote that the execution of the alternatives timed out.

7 Conclusions

In this paper, we have presented a framework for data integration entirely based on graphs. In the proposed approach all classical constructs such as schema, queries and mappings are represented using graphs. We advocate that such unique, and widely accepted, data management formalism allows non-technical users to perform exploratory tasks, such as data wrangling. On top of that, the flexibility of graphs enables the extensibility of the current rewriting algorithm. For example, to jointly consider aggregations when running the rewriting algorithm. We have additionally presented solid foundations for the design of rewriting algorithms that preserve desired query containment properties Our experimental results show that there is no significant overhead for join discovery, and that, as usual on algorithms for answering queries using views, the major source of complexity is the number of data sources. Despite this, and the fact that our rewriting semantics are richer, the performance of our approach is superior to that of alternative methods for answering queries using views.

Acknowledgements. This work is partly supported by Barcelona's City Council under grant agreement 20S08704.

References

[1] S. Kandel, J. Heer, C. Plaisant, J. Kennedy, F. van Ham, N. H. Riche, C. Weaver, B. Lee, D. Brodbeck, and P. Buono, "Research directions in data wrangling: Visualizations and transformations for usable and credible data," *Information Visualization*, vol. 10, no. 4, pp. 271–288, 2011.

[2] M. Buoncristiano, G. Mecca, E. Quintarelli, M. Roveri, D. Santoro, and L. Tanca, "Database challenges for exploratory computing," *SIGMOD Record*, vol. 44, no. 2, pp. 17–22, 2015.

[3] R. Angles and C. Gutiérrez, "Survey of graph database models," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 1:1–1:39, 2008.

[4] N. Konstantinou, E. Abel, L. Bellomarini, A. Bogatu, C. Civili, E. Irfanie, M. Koehler, L. Mazilu, E. Sallinger, A. A. A. Fernandes, G. Gottlob, J. A. Keane, and N. W. Paton, "VADA: an architecture for end user informed data preparation," *J. Big Data*, vol. 6, p. 74, 2019.

[5] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, and G. Xiao, "Ontop: Answering SPARQL queries over relational databases," *Semantic Web*, vol. 8, no. 3, pp. 471–487, 2017.

[6] T. Furche, G. Gottlob, L. Libkin, G. Orsi, and N. W. Paton, "Data wrangling for big data: Challenges and opportunities," in *EDBT*, 2016.

[7] B. Golshan, A. Y. Halevy, G. A. Mihaila, and W. Tan, "Data integration: After the teenage years," in *PODS*, 2017.

[8] M. Stonebraker and I. F. Ilyas, "Data integration: The current status and the way forward," *IEEE Data Eng. Bull.*, vol. 41, no. 2, pp. 3–9, 2018.

[9] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev, "Reasoning over extended ER models," in *ER*, 2007.

[10] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom, "The TSIMMIS approach to mediation: Data models and languages," *J. Intell. Inf. Syst.*, vol. 8, no. 2, pp. 117–132, 1997.

[11] M. T. Roth, M. Arya, L. M. Haas, M. J. Carey, W. F. Cody, R. Fagin, P. M. Schwarz, J. Thomas, and E. L. Wimmers, "The Garlic Project," in *SIGMOD*, 1996.

[12] D. Beneventano, S. Bergamaschi, S. Castano, A. Corni, R. Guidetti, G. Malvezzi, M. Melchiori, and M. Vincini, "Information integration: The MOMIS project demonstration," in *VLDB*, 2000.

[13] A. Y. Levy, A. Rajaraman, and J. J. Ordille, "Querying heterogeneous information sources using source descriptions," in *VLDB*, 1996.

[14] O. M. Duschka, M. R. Genesereth, and A. Y. Levy, "Recursive query plans for data integration," *J. Log. Program.*, vol. 43, no. 1, pp. 49–73, 2000.

[15] R. Pottinger and A. Y. Halevy, "Minicon: A scalable algorithm for answering queries using views," *VLDB Journal*, vol. 10, no. 2-3, pp. 182–198, 2001.

[16] M. Arenas, P. Barceló, L. Libkin, and F. Murlak, *Foundations of Data Exchange*. Cambridge University Press, 2014.

[17] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi, "Query processing under GLAV mappings for relational and graph databases," *Proc. VLDB Endow.*, vol. 6, no. 2, pp. 61–72, 2012.

[18] F. Priyatna, Ó. Corcho, and J. F. Sequeda, "Formalisation and experiences of r2rml-based SPARQL to SQL query translation using morph," in *WWW*, 2014.

[19] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo, "The MASTRO system for ontology-based data access," *Semantic Web*, vol. 2, no. 1, pp. 43–53, 2011.

[20] C. Beeri, A. Y. Levy, and M. Rousset, "Rewriting queries using views in description logics," in *PODS*, 1997.

[21] F. Goasdoué and M. Rousset, "Answering queries using views: A KRDB perspective for the semantic web," *ACM Trans. Internet Techn.*, vol. 4, no. 3, pp. 255–288, 2004.

[22] J. Baget, M. Leclère, M. Mugnier, S. Rocher, and C. Sipieter, "Graal: A toolkit for query answering with existential rules," in *RuleML*, 2015.

[23] M. Buron, F. Goasdoué, I. Manolescu, and M. Mugnier, "Ontology-based RDF integration of heterogeneous data," in *EDBT*, 2020.

[24] S. S. Bhowmick, B. Choi, and C. Li, *Human Interaction with Graphs: A Visual Querying Perspective*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2018.

[25] S. Nadal, O. Romero, A. Abelló, P. Vassiliadis, and S. Vansummeren, "An integration-oriented ontology to govern evolution in big data ecosystems," *Inf. Syst.*, vol. 79, pp. 3–19, 2019.

[26] A. P. Sheth and J. A. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," *ACM Comput. Surv.*, vol. 22, no. 3, pp. 183–236, 1990.

[27] M. Lenzerini, "Data integration: A theoretical perspective," in *PODS*, 2002.

[28] A. Y. Halevy, "Answering queries using views: A survey," *VLDB J.*, vol. 10, no. 4, pp. 270–294, 2001.

[29] G. Konstantinidis and J. L. Ambite, "Scalable query rewriting: a graph-based approach," in *SIGMOD*, 2011.

[30] ——, "Optimizing the chase: Scalable data integration under constraints," *PVLDB*, vol. 7, no. 14, pp. 1869–1880, 2014.

[31] M. Benedikt, G. Konstantinidis, G. Mecca, B. Motik, P. Papotti, D. Santoro, and E. Tsamoura, "Benchmarking the chase," in *PODS*, 2017.

[32] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati, "Linking data to ontologies," *Journal on Data Semantics*, vol. 10, pp. 133–173, 2008.

[33] G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati, "Using ontologies for semantic data integration," in *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*, ser. Studies in Big Data, 2018, vol. 31, pp. 187–202.

[34] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, and M. Ruzzi, "Using OWL in data integration," in *Semantic Web Information Management*, 2009, pp. 397–424.

[35] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[36] A. Bonifati, G. H. L. Fletcher, H. Voigt, and N. Yakovets, *Querying Graphs*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2018.

[37] D. Maier, A. O. Mendelzon, and Y. Sagiv, "Testing implications of data dependencies," *ACM Trans. Database Syst.*, vol. 4, no. 4, pp. 455–469, 1979.

[38] R. M. Karp, "On the computational complexity of combinatorial problems," *Networks*, vol. 5, no. 1, pp. 45–68, 1975.

[39] S. Nadal, K. Rabbani, O. Romero, and S. Tadesse, "ODIN: A dataspace management system," in *ISWC*, 2019.

Sergi Nadal received the PhD in Computer Science in 2019 from Universitat Politècnica de Catalunya (UPC) and Université Libre de Bruxelles (ULB). He is a postdoctoral fellow and teaching assistant in the Database Technologies and Information Management (DTIM) group in UPC. His research interests are on systems aspects on data and information management.

Alberto Abelló is an associate professor at Departament d'Enginyeria de Serveis i Sistemes d'Informació (ESSI) of Universitat Politècnica de Catalunya. PhD in Informatics, UPC. Local coordinator of the Erasmus Mundus PhD program IT4BI-DC. Active researcher with more than 100 peer-reviewed publications and H-factor of 28, his interests include Data Warehousing and OLAP, Ontologies, NOSQL systems and Big Data management.

Oscar Romero is an associate professor at Departament d'Enginyeria de Serveis i Sistemes d'Informació (ESSI) of Universitat Politècnica de Catalunya. Local coordinator of the Erasmus Mundus in Big Data Management and Analytics (BDMA) programme and the Data Science master at the Faculty of Informatics of UPC. His main interests are data management, data integration and data-intensive flows.

Stijn Vansummeren is a research professor in data management and data wrangling at the Data Science Institute of Hasselt University, Belgium His research interests are in data management viewed broadly, where he focuses on both foundational and systems aspects. Most recently, he has worked on dynamic query processing, information extraction, data integration, and structural indexes.

Panos Vassiliadis is a professor at the University of Ioannina, Greece. His research focuses on the rigorous modeling of data, software, and their interdependence. Currently he works in the areas of business intelligence and schema evolution. He is a senior member of the IEEE. More information is available at http://www.cs.uoi.gr/~pvassil.