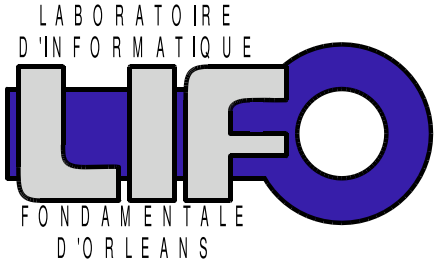
Querying Semantic Graph Databases in View of Constraints and Provenance

Jacques Chabin, Mirian Halfeld Ferrari, Thanh Binh Nguyen



4 rue Léonard de Vinci
BP 6759
F-45067 Orléans Cedex 2
FRANCE
http://www.univ-orleans.fr/lifo

Rapport de Recherche

Querying Semantic Graph Databases in View of Constraints and Provenance

Jacques Chabin, Mirian Halfeld Ferrari, Thanh Binh Nguyen
LIFO, Université d'Orléans

Rapport n^o **RR-2016-02**

Querying Semantic Graph Databases in View of Constraints and Provenance*

Jacques Chabin, Mirian Halfeld Ferrari, and Thanh Binh Nguyen**

Université Orléans, INSA CVL - LIFO EA, Orléans, France
{jchabin, mirian, binh}@univ-orleans.fr

Abstract. This paper focus on query semantics on a graph database with constraints defined over a global schema and confidence scores assigned to the local data sources (which compose a distributed graph instance). Our query environment is customizable and user-friendly: it is settled on the basis of data source confidence and user's quality restrictions. These constraints are on queries not on sources. Inconsistent data is filtered, allowing the production of valid results even when data quality cannot be entirely ensured by sources. We elaborate on the process of validating answers to the queries against positive, negative and key constraints. Our validator can interact with divers lower-level query evaluation methods.

Keywords: semantic data graph, constraint, personalization, provenance

1 Introduction

Query answering is more challenging when dealing with multiple data sources having different confidence degrees, incomplete or inconsistent data. One may also expect query answering to be ontology-mediated over data sets coupled with inference reasoning. Nowadays, data collections are expected to have these characteristics ([8]), which are important aspects also for the big data research. In this modern scenario, constraint verification are generally neglected due to their cost; the lack of data quality (Veracity) and the rapidity of data changes (Velocity) – two of the Vs ascribed to big data challenges – being difficult obstacles to overcome. Moreover, the need of powerful languages to bolster the increase demand of analysis over graphs and networks (as in [10,22]) led researchers to focus on user-friendly platforms capable of dealing with large-scale applications without neglecting quality.

Constraints are the expression of the desired *answer quality* and of a *fixed (or personalised) context*. This paper considers them as first-class citizens again by introducing a method to constrain query answers instead of data sources. They settle the validity requirements a user is looking for. Constraint verification is

* Work partially supported by APR-IA GIRAFON.
** Supported by a PhD grant Orléans-Tours.

triggered by the query components (the query body). Inconsistency on sources is allowed but query answers are filtered to ensure consistency *w.r.t.* constraints (an original proposal towards big data applications).

The goal of this paper is to propose a declarative (datalog-type) query language over a semantic graph database together with a mechanism for filtering answers according to a customised context that settles global constraints and confidence degrees. Our short-term objective is to deal with ontologies or data sets seen as a property graph instance. Our long-term goal is a user-friendly environment to query big amount of distributed data for further (mining) analysis. The extension of our query language to deal with *group by* and some aggregate functions is another step towards this direction. Even if the paper mentions our basic assumptions concerning data sources, its focus is on the definition of the semantics of query answers on the proposed global environment. Details concerning query evaluation are out of its scope.

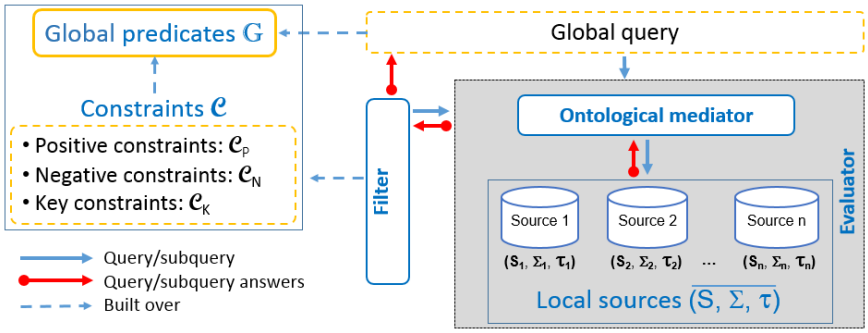


Fig. 1. Query system overview

General System Architecture. We deal with a graph database system whose main parts are illustrated in Figure 1. A *global system* offers a *graph schema* composed by *global predicates* and *constraints*. Constraints are special datalog rules over global predicates. A *local system* (composed by different source databases) stores a *distributed instance of the graph*. In a general perspective, we consider that each *local source* has explicitly stored data and may have *inference rules* allowing the derivation of new data from stored one. Inference rules are linear tuple generated dependencies (linear TGD), expressed in Datalog$^{\pm}$ [6]. We consider datalog *queries*, posed on the global system, and evaluated over the distributed database. This evaluation is performed by a (lower-level) *query evaluator* for which different mechanisms can be considered. In this paper we offer some discussion on these possibilities, but we focus on the higher-level of our system and consider the evaluator as a module that receives the query and renders answers whose

quality is to be checked. Indeed, the query body may trigger constraints established at the global level and, in this case, query answers are filtered according to them. We assume, as part of the valuator, the existence of an *ontological mediator* which ensures the communication between the global and the local systems. This mediator dispatches sub-queries to local sources and computes responses to a global query from sub-query answers. Constraint verification may generate auxiliary simpler queries. Furthermore, sources are not trusted equally. Each data source has a *confidence degree*, settled by a user or proposed by the system. The query answering process takes them into account to compose responses.

Paper Organization. This paper offers: a motivation example (Section 2); main definitions (Sections 3 and 4); the semantics of constrained queries over sources with different confidence degrees (Section 5); implementation aspects (Section 6); a language extension, related work and conclusions (Sections 7, 8 and 9).

2 Motivating example

In a global schema concerning research production we consider as global predicates: $Journal(X_@)$ and $Conf(X_@)$ classify the publications into journals or conference proceedings; $Ranking(X_@, X_r)$ indicates their ranking; $Prod(X_t, X_a, X_y, X_@, X_l)$ lists the production of a research laboratory (X_l) with the paper title (X_t), author (X_a), year (X_y) and where it is published ($X_@$); $Aff(X_a, X_l, X_y)$ refers to authors affiliation and $CNRS(X_l)$ and $Univ(X_l)$ indicate possible financial support of French laboratories. Constraints over these global predicates are shown in Table 1.

$\mathcal{C}_P$	POSITIVE CONSTRAINTS
c_{P_1}	$: Prod(X_t, X_a, X_y, X_@, X_l) \rightarrow Aff(X_a, X_l, X_y)$
c_{P_2}	$: Aff(X_a, X_l, X_y) \rightarrow CNRS(X_l)$
c_{P_3}	$: Prod(X_t, X_a, X_y, X_@, X_l) \rightarrow Journal(X_@)$
$\mathcal{C}_N$	NEGATIVE CONSTRAINTS
c_{N_1}	$Conf(X_@), Ranking(X_@, \mathbf{C}) \rightarrow \bot$
c_{N_2}	$Ranking(X_@, \mathbf{C}) \rightarrow \bot$
$\mathcal{C}_K$	KEY CONSTRAINTS
c_{K_1}	$Aff(X_a, X_l, X_y), Aff(X_a, Y_l, X_y) \rightarrow X_l = Y_l$
c_{K_2}	$Ranking(X_@, X_r), Ranking(X_@, Y_r) \rightarrow X_r = Y_r$

Table 1. Set of constraints on $\mathbb{G}$

A first user wants to work in a context that contains only constraints c_{P_1} (an author's production in the laboratory production list implies that the author is affiliated to this lab), c_{N_1} (no conference ranked **C** is taken into account), c_{K_1} (authors cannot be affiliated to different laboratories at the same time) and c_{K_2} (publications cannot have different rankings).

Queries are built over this global (personalised) schema, and their results are computed from data coming from a distributed database, composed by our so called local data sets. Figures 2-3 illustrate local sources which are not trusted equally by the global system. A confidence degree (τ) indicates the accuracy associated to each one. With these source confidence degrees the entire querying context is settled. Source 1 is considered to be accurate (95% reliable) while the reliance on Source 4 is smaller (accuracy: 65%). Sources can have inference rules. For instance, Sources 1 and 3 infer the existence of lab affiliations for a person being a researcher (Res) or a professor ($Prof$) at a university, respectively.

Source 1, $\tau_{S1} = 0.95$
$ProdS1(T1, Bob, 2014, TODS, L1)$
$ProdS1(T2, Bob, 2015, ODBASE, L2)$
$ProdS1(T2, Tom, 2015, ODBASE, L2)$
$AffS1(Bob, L2, 2015)$
$CNRSS1(L2)$
$UnivS1(L3)$
$UnivS1(L1)$
$ProdS1(T5, Tom, 2015, TLDKS, L2)$
$ProdS1(T6, Sue, 2016, TLDKS, L2)$
$Res(Sue, 2016, Orleans, 1)$
$Res(Alice, 2016, Paris, 1)$
$\Sigma_1 : Res(X_a, X_y, X_{univ}, X_{Cl}) \rightarrow \exists X_l Aff(X_a, X_l, X_y)$

Source 2, $\tau_{S2} = 0.75$
$RankingS2(TODS, A)$
$RankingS2(ICEIS, C)$
$JournalS2(TODS)$
$ConfS2(ICEIS)$
$ConfS2(ODBASE)$
$JournalS2(TLDKS)$
$RankingS2(TLDKS, A)$

Fig. 2. Example of local sources

In this scenario let us consider query

$q_1(X_l, X_@, X_r) \leftarrow Prod(X_t, X_a, X_y, X_@, X_l), Conf(X_@), Ranking(X_@, X_r)$

to find the conferences appearing in a laboratory production list and their rankings. The required confidence degree is $\tau_{in} = 0.6$, indicating that sources having a smaller confidence degree should not be taken into account. The answer is the set $\{(L2, ODBASE, B) : 0.65\}$ as we have $ProdS1(T2, Tom, 2015, ODBASE, L2)$ (which triggers c_{P_1} giving $AffS3(Tom, L2, 2015)$), $ConfS2(ODBASE)$ and $RankingS4(ODBASE, B)$. Tuple $(L3, ICEIS, C)$ is not an answer since different constraints are violated. Even if two productions exist in $ICEIS$, author $Mary$ is affiliated to two labs in year 2016 (violation of c_{K_1}) while Joe has no affiliation (violation of c_{P_1}). Moreover, $ICEIS$ has ranking **C** (violation of c_{K_2}).

A second stricter user wants to change the context to a new one where all the constraints in Table 1 are used. Now, only journal publication (c_{P_3}), not ranked **C** (c_{N_2}), concerning a CNRS lab (c_{P_2}) are considered. In this context let us consider the query

$$q_2(X_l, X_@, X_r, X_t) \leftarrow Prod(X_t, X_a, X_y, X_@, X_l), Ranking(X_@, X_r)$$

with required confidence degree $\tau_{in} = 0.75$. The answer is $\{(L1, TODS, A, T1) : 0.75, (L2, TLDKS, A, T5) : 0.75\}$. Source 4 is not considered since $\tau_{S4} < \tau_{in}$, avoiding the violation of constraints c_{K_2} and c_{N_2}.

Source 3, $\tau_{S3} = 0.85$
$ProdS3(T1, Bob, 2014, TODS, L1)$
$ProdS3(T3, Mary, 2016, ICEIS, L3)$
$ProdS3(T4, Joe, 2016, ICEIS, L3)$
$AffS3(Bob, L1, 2014)$
$AffS3(Tom, L2, 2015)$
$AffS3(Mary, L3, 2016)$
$AffS3(Mary, L1, 2016)$
$CNRSS3(L1)$
$Prof(Mary, 2014, Orleans)$
$Prof(Ann, 2015, Tours)$
$\Sigma_3 : Prof(X_a, X_y, X_{univ}) \rightarrow \exists X_l Aff(X_a, X_l, X_y)$

Source 4, $\tau_{S4} = 0.65$
$RankingS4(TODS, B)$
$RankingS4(ODBASE, B)$
$RankingS4(ICEIS, C)$
$RankingS4(TLDKS, C)$

Fig. 3. Example of local sources (cont.)

3 A graph database

A graph schema $\mathbb{G}$ (or simply schema), is a finite set of predicate symbols or relation names $G_1 \dots G_m$ associated to a set of constraints $\mathcal{C}$ defined over $\mathbb{G}$. Queries are built over the global schema but evaluated over local sources. Each answer is verified *w.r.t.* $\mathcal{C}$. Data remain in sources and are obtained when local systems are queried. Local sources store data explicitly or implicitly via inference rules which are capable of deriving new data from stored one. Let $\mathbb{S}_1, \dots, \mathbb{S}_n$ be the local schemas of n source databases $(\mathcal{S}_i, \Sigma_i, \tau_i)$, where $\mathcal{S}_i$ is a data source instance, Σ_i the associated set of inference rules defined over $\mathbb{S}_i$ $(1 \leq i \leq n)$ and τ_i is the source confidence degree, represented by a number in the interval $[0, 1]$.

Alphabet and atomic formulas. Let **A** be an alphabet consisting of constants, variables, predicates, the equality symbol (=), quantifiers ($\forall$ and $\exists$) and the symbols $\top$ (true) and $\bot$ (false). We consider four mutually disjoint sets, namely: (1) Δ_C, a countably infinite set of constants, called the underlying database domain; (2) Δ_N, a countably infinite set of fresh labelled nulls which are placeholders for unknown values; (3) VAR an infinite set of variables used to range over elements of $\Delta_C \cup \Delta_N$ and (4) PRED, a *finite* set of predicates or relation names (each predicate is associated with a positive integer called its arity). The only possible terms are constants, nulls or variables. An atomic formula (or atom) has one of the forms: (i) $P(t_1, ..., t_n)$, where P is an n-ary predicate, and $t_1, ..., t_n$ are terms; (ii) expressions $\top$ (true) and $\bot$ (false) or (iii) $t_1 = t_2$ (where t_1 and t_2 are terms). A conjunction of atoms is often identified with the set of all its atoms. We denote by **X** sequences of terms $X_1 \dots X_k$ where $k \geq 0$ (in the context one can understand when only variables are used).

Substitution. A *substitution* from one set of symbols E_1 to another set of symbols E_2 is a function $h : E_1 \Rightarrow E_2$. A *homomorphism* from a set of atoms A_1 to a set of atoms A_2, both over the same schema R, is a substitution h from the set of terms of A_1 to the set of terms of A2 such that: (i) if $t \in \Delta_C$, then $h(t) = t$, and

(ii) if $r(t_1, ..., t_n)$ is in A_1, then $h(r(t_1, ..., t_n)) = r(h(t_1), ..., h(t_n))$ is in A_2. The notion of homomorphism naturally extends to conjunctions of atoms.

3.1 Source databases.

A database schema $\mathbb{S}$ is composed by a finite set of predicate symbols and a (optional) set of inference rules Σ. The models of $\mathcal{S}$ *w.r.t.* Σ, denoted as $mods(\mathcal{S}, \Sigma)$, is the set of all instances I such that $I \models \mathcal{S} \cup \Sigma$, which means that $I \supseteq \mathcal{S}$ and I satisfies Σ.

An *instantiated atom* over a predicate $S \in \mathbb{S}$ is an expression of the form $S(u)$ where $u \in (\Delta_C \cup \Delta_N)^n$. A *fact* (or a ground atom) over $S \in \mathbb{S}$ is an instantiated atom with no null values (*i.e.*, atoms of form $S(u)$ where $u \in (\Delta_C)^n$). Under the logic-programming perspective, an instance over S is a finite set of instantiated atoms over S while an instance over schema $\mathbb{S}$ is a finite set $\mathcal{S}$ which is the union of instances over S, for all $S \in \mathbb{S}$.

Inference rules are tuple-generating dependency, TGD (denoted as basic Datalog$^{\pm}$ rules). A TGD σ over a database schema $\mathbb{S}$ is a first-order formula $\forall \mathbf{X},\mathbf{Y}\ \phi(\mathbf{X},\mathbf{Y}) \rightarrow \exists\ \mathbf{Z}\ \psi(\mathbf{X},\mathbf{Z})$ where $\phi(\mathbf{X},\mathbf{Y})$ and $\psi(\mathbf{X},\mathbf{Z})$ are conjunctions of atoms over $\mathbb{S}$, called the body and the head of σ and denoted $body(\sigma)$ and $head(\sigma)$, respectively. Usually, we omit the universal quantifiers. The chase (a fundamental algorithmic tool introduced for checking implication of dependencies) works on an instance through the so-called *TGD chase rule* ([12]): Given schema $\mathbb{S}$, let $\mathcal{S}$ be an instance and $\sigma : \forall \mathbf{X},\mathbf{Y}\ \phi(\mathbf{X},\mathbf{Y}) \rightarrow \exists\ \mathbf{Z}\ \psi(\mathbf{X},\mathbf{Z})$ a TGD over $\mathbb{S}$. If σ is applicable to $\mathcal{S}$, *i.e.*, there is a homomorphism h such that $h(\phi(\mathbf{X}, \mathbf{Y})) \subseteq \mathcal{S}$ then (i) define $h' \supseteq h$ such that $h'(Z_i) = z_i$, for each $Z_i \in \mathbf{Z}$, where $z_i \in \Delta_N$ is a fresh labelled null not introduced before, and (ii) add to $\mathcal{S}$ the set of atoms in $h'(\psi(\mathbf{X},\mathbf{Z}))$, if not already in $\mathcal{S}$.

Given a source database $(\mathcal{S}, \Sigma)$, the chase algorithm ([6]) completes the database by an exhaustive application of the TGD chase rule which leads to a possible infinite result denoted by $chase(\mathcal{S}, \Sigma)$. The chase for $(\mathcal{S}, \Sigma)$ is a universal model of $\mathcal{S}$ *w.r.t.* Σ, *i.e.*, for each $I \in mods(\mathcal{S}, \Sigma)$, there exists a homomorphism from $chase(\mathcal{S}, \Sigma)$ to I.

For each local source, the set Σ contains only linear TGD. A *guarded* TGD ([6,12]) has an atom in its body which contains all the universally quantified variables. A guarded TGD is *linear* iff it contains only one atom in its body. Query answering under linear TGD is highly tractable ([6]).

Example 1. In Figure 3, Source 3 has the inference rule $Prof(X_a, X_y, X_{univ}) \rightarrow \exists X_l Aff(X_a, X_l, X_y)$ with which it is possible to infer $Aff(Mary, z_1, 2014)$ and $Aff(Ann, z_2, 2015)$ where z_1 and z_2 are fresh nulls. □

3.2 Global schema.

A global schema $\mathbb{G}$ serves as the unique entry point on which global queries are posed by users ([3]). The global schema (that may be visualised by a property

graph) is composed by a set of global relation names $G_1 \dots G_n$ and a set of constraints $\mathcal{C}$ defined over $\mathbb{G}$. An instance for $\mathbb{G}$ is distributed on divers sources (information in each data source corresponds to node or edge instances).

Constraints are restrictions imposed on queries; only data respecting them are allowed as query answers. The originality of our approach is due to the following aspects: (*i*) A constraint violation implies discarding only invalid answers (not the whole database). (*ii*) The distributed database (local system) may contain non valid data (perhaps derived by a tgd) which are filtered when involved in a global query. (*iii*) Constraints are active rules imposing a dialogue between the global and the local levels: on the global level, they are triggered by facts in the query body (facts resulting from query instantiation, during evaluation). Triggered constraints usually impose extra verifications on the local level (*e.g.* testing if another fact is true in the distributed database).

A set $\mathcal{C}$ of constraints over $\mathbb{G}$ (with global relation names A and B) is composed by three subsets, as follows:

(1) Positive constraints ($\mathcal{C}_P$): Each positive constraint has the form

$$\forall \mathbf{X}, \forall \mathbf{Y}\ A(\mathbf{X}) \rightarrow B(\mathbf{Y}) \text{ with } \mathbf{Y} \subseteq \mathbf{X}.$$

Positive constraints cover inclusion constraints, but also more powerful ones such as $A(X) \rightarrow B(X, X)$. They are similar to the update rules in [13]; they act as active rules that generate the facts that must be true in the distributed database.

(2) Negative constraints ($\mathcal{C}_N$): Each negative constraint has the form

$$\forall \mathbf{X}\ \phi(\mathbf{X}) \rightarrow \bot$$

where $\phi(\mathbf{X})$ is an atom $A(\mathbf{X})$ or a conjunction of two atoms $A(\mathbf{X}_1), B(\mathbf{X}_2)$, having a non-empty intersection between the terms in $\mathbf{X}_1$ and in $\mathbf{X}_2$.

(3) Functional dependency constraints ($\mathcal{C}_K$) also called key constraints: Each functional dependency is an equality-generating dependency (or EGD), without nulls, having the form:

$$A(\mathbf{Y}, X_1, \mathbf{Z}_1), A(\mathbf{Y}, X_2, \mathbf{Z}_2) \rightarrow X_1 = X_2$$

where $\mathbf{Y}$ is a sequence having at least one term.

In all kinds of constraints, the left-hand side is called the body of the constraint and consists of one atom or a conjunction of two atoms. We refer to $body(c)$ as the set of atoms in the body of a given constraint c. The right-hand side of a constraint c is its head. The head can have one of the following atomic formulas: a positive atom (for $\mathcal{C}_P$), the atom $\bot$ (for $\mathcal{C}_N$) or an atom with the equality symbol $X_1 = X_2$ (for $\mathcal{C}_K$). It is worth noting that a constraint is activated only when its body is instantiated by a fact.

Example 2. In Section 2, *Joe* is not affiliated to a lab. The inconsistency of the distributed database *w.r.t.* c_{P_1} is tolerated, but inconsistent information is discarded from q_1 answer. □

When the body of a positive constraint matches a ground atom, the (now instantiated) constraint head triggers side effects, setting other facts which should

also be true in the distributed database. To formally define this behaviour, we introduce an immediate consequence operator.

Definition 1 (Immediate consequence operator). Let T be an operator over a given set of constraints in $\mathcal{C}$. Given an instance I over $\mathbb{G}$, and a constraint c, let ν be a homomorphism that instantiates $body(c)$ with constants appearing in I. Let $T_{\mathcal{C}}(I) = I \cup \{\nu(head(c)) \mid c \in \mathcal{C} \text{ and } \nu(body(c)) \subseteq I\}$. When no doubts are possible, we simply write T instead of $T_{\mathcal{C}}$. □

In the above definition, when $\mathcal{C}$ is replaced by $\mathcal{C}_P$, results in [13] can be applied, *i.e.* T is monotonic, the sequence defined by: $T^0(I) = I$ and $T^k(I) = T(T^{k-1}(I))$, for every integer $k > 0$, has a limit. This limit, referred to as the *least fixed point of* T *w.r.t.* I, is denoted by $lfp(T, I)$ or by $T^*(I)$.

Example 3. Consider the second context of Section 2 and $I = \{Prod(T, John, 2016, VLDB, L1)\}$. Clearly we have $T^*(I) = \{Prod(T, John, 2016, VLDB, L1), Aff(John, L1, 2016), CNRS(L1)\}$. In our query environment I is initialized by the body instantiations obtained during the query evaluation. □

4 Queries

We recall the general definition of conjunctive queries.

Definition 2 (Conjunctive query). A *conjunctive query* (CQ) q of arity n over a given schema is a formula of the form $q(\mathbf{X}) \leftarrow \phi(\mathbf{X},\mathbf{Y})$, where $\phi(\mathbf{X},\mathbf{Y})$, called the body of q, is a conjunction of atoms over the schema and $q(\mathbf{X})$, denoted as the head of q is an n-ary predicate. A *boolean conjunctive* query (BCQ) is a CQ of arity zero. We denote by $body(q)$ (respect. $head(q)$) the set of atoms composing the body (respect. the head) of a given query q.

Let I be an instance for the given schema. The *answer* to a CQ q of arity n over I, denoted as $q(I)$, is the set of all n-tuples $t \in (\Delta_C)^n$ for which there exists a homomorphism $h_t : X \cup Y \Rightarrow \Delta_C \cup \Delta_N$ such that $h_t(\phi(\mathbf{X},\mathbf{Y})) \subseteq I$ and $h_t(\mathbf{X}) = t$. We denote by h_t *a homomorphism used to obtain an answer tuple* t.

Technically, the answer *false* (*i.e.*, a negative answer) for a BCQ corresponds to the empty result set and the answer *true* (*i.e.*, a positive answer) corresponds to the result set containing the empty tuple. A positive answer over I is denoted by $I \models q$. A union of CQ (UCQ) Q of arity n is a set of CQ, where each $q \in Q$ has the same arity n and uses the same predicate symbol in the head. The answer to Q over an instance I, denoted as $Q(I)$, is defined as the set of tuples $\{t \mid$ there exists $q \in Q$ such that $t \in q(I)\}$. □

Query answers over a local database are computed by taking into account stored facts and inference rules.

Definition 3 (Query answering under TGD). Let $(\mathcal{S}, \Sigma)$ be a given local database. The models of $\mathcal{S}$ *w.r.t.* Σ, denoted as $mods(\mathcal{S}, \Sigma)$, is the set of all instances I such that $I \models \mathcal{S} \cup \Sigma$, which means that $I \supseteq \mathcal{S}$ and I satisfies Σ. The answer to a CQ q *w.r.t.* $\mathcal{S}$ and Σ, denoted as $ans(q, (\mathcal{S}, \Sigma))$, is the set $\{t \mid t \in q(I)$ for each $I \in mods(\mathcal{S}, \Sigma)\}$. The answer to a BCQ q *w.r.t.* $\mathcal{S}$ and Σ is positive, denoted as $(\mathcal{S} \cup \Sigma) \models q$, iff $ans(q, (\mathcal{S}, \Sigma)) \neq \emptyset$. □

Example 4. We consider the instance $(\mathcal{S}_3, \Sigma_3)$ of **Source** 3 (Figure 3). The evaluation of $q_a(X_a) \leftarrow Aff S3(X_a, X_l, 2015)$ over this instance gives $ans(q_a, (\mathcal{S}_3, \Sigma_3)) = \{(Tom), (Ann)\}$. Tuple (Ann) results from the inference available in Σ_3. □

In this paper, we set aside aspects of mapping global and local systems (see [3] for an overview) and assume that when a global query q has a non-empty set of answers then there exists at least one re-writing of q in terms of sub-queries $q'_1 \dots q'_m$ where each q'_j $(1 \leq j \leq m)$ is a sub-query to be evaluated in the local database $(\mathcal{S}_{i_j}, \Sigma_{i_j}))$ $(1 \leq i_j \leq n)$.

Our approach can be presented as an independent module that interacts with a *query evaluation machine over a distributed database*. It sends the query to this machine, treats and filters its results, possibly with the help of simpler queries. Different querying mechanisms can be envisaged as we will discuss in Section 8.

We write $\overline{(\mathcal{S}, \Sigma)} \models q$, a shorthand of $(\mathcal{S}_1, \Sigma_1) \sqcup \dots \sqcup (\mathcal{S}_n, \Sigma_n) \models q$, to denote that the answer of a BCQ q is positive *w.r.t.* to all local databases. In this way, we see local databases as a whole, *i.e.*, a local system (or distributed database) capable of answering our global query. Notice the use of a disjoint union to express that inferences work locally. Each source contributes to the global answer with all its (inferred or stored) data. We denote by $ans(q, \overline{(\mathcal{S}, \Sigma)})$ the set of tuples obtained as answers for a conjunctive global query q over a distributed database $\overline{(\mathcal{S}, \Sigma)}$. For instance, from Example 4, $ans(q_a, \overline{(\mathcal{S}, \Sigma)}) = \{(Bob), (Tom), (Ann)\}$.

5 Constrained queries within a confidence degree

To produce trustable answers, the global system assigns a confidence degree τ_i to each data source $(\mathcal{S}_i, \Sigma_i)$ and assume that a query q is associated to a minimal confidence degree τ_{in}. Only information coming from source databases whose confidence degrees respect a given condition *w.r.t.* τ_{in} should be taken into account to build answers for q.

Definition 4 (Local querying with confidence). Let $(\mathcal{S}, \Sigma, \tau)$ be a local source database where $\mathcal{S}$ is a database instance, Σ is the associated set of inference rules and τ is the truth or confidence degree of the database. Let q be query over $(\mathcal{S}, \Sigma, \tau)$ with the minimum required truth degree τ_{in}. The answer of this query over $(\mathcal{S}, \Sigma, \tau)$ is defined as follows:

$$ans(q{:}\tau_{in}, (S, \Sigma, \tau)) = \{(t : \tau_{out}) \mid \tau_{out} = \tau \text{ and } cond(\tau_{in}, \tau_{out}) \text{ and } t \in q(I) \text{ for each } I \in mods(\mathcal{S}, \Sigma)\}.$$

where $cond(\tau_{in}, \tau_{out})$ is a condition we may establish to avoid considering some sources. □

We recall that in our examples in this paper, we use $cond(\tau_{in}, \tau_{out}) = (\tau_{out} \geq \tau_{in})$ discarding all sources whose confidence is inferior to τ_{in}. One may decide to settle no condition at this step, taking into account all sources in the computation of a tuple confidence degree.

Now, we consider how to put together answers produced by differently trusted database, in order to compose the response for a given query $q{:}\tau_{in}$. To this end, we build a set of possible candidate answers.

Definition 5 (Candidate answer over $\overline{(\mathcal{S}, \Sigma, \tau)}$). Let $\overline{(\mathcal{S}, \Sigma, \tau)}$ be a graph database instance composed of n local databases having different truth degrees. A couple $(t : \tau_{out})$ is a candidate answer for a global query $(q : \tau_{in}, \overline{(\mathcal{S}, \Sigma, \tau)})$ if the following conditions hold:

1. t is a tuple such that $t \in ans(q, \overline{(\mathcal{S}, \Sigma)})$;
2. $\tau_{in} \leq \tau_{out}$ and
3. $\tau_{out} = f(\tau_{in}, \{\tau^1_{out_{S_i}}, \ldots, \tau^m_{out_{S_l}}\})$ where:
   - each $\tau^k_{out_{S_j}}$ is the degree of the tuples in $ans(q_k : \tau^k_{in}, (S_j, \Sigma_j, \tau_j))$ for the sub-query q_k $(1 \leq k \leq m)$ generated to be evaluated on the local source (S_j, Σ_j, τ_j) during the evaluation process of q (where i, j, l are integers in $[1, n]$) and
   - f is a function which computes a confidence degree from the query confidence degree and the set of confidence degrees of data sources concerned by the query. □

It is worth noting that different functions f can be used; allowing some flexibility to the user who can parametrize the use of confidence degrees. Let us consider that in Definition 4 $cond(\tau_{in}, \tau_{out}) = true$. In this case, the selection of t is based *only* on the confidence degree computed by f – for instance, when f is the average of all source confidence degrees, resulting τ_{out} may take in account data sources whose confidence is less than τ_{in}. If however a condition such as $\tau_{out} \geq 0.5$ is used in Definition 4, only sources respecting it are used in the average computation. In this paper, our examples consider that $f(\tau_{in}, \{\tau^1_{out_{S_i}}, \ldots, \tau^m_{out_{S_l}}\})$ corresponds to $min(\{\tau^1_{out_{S_i}}, \ldots, \tau^m_{out_{S_l}}\})$, and as stated above, we disregard sources whose confidence is inferior to τ_{in}.

Example 5. In Section 2, q_1 answer $(L2, ODBASE, B)$ with $\tau_{out} = 0.65$ may be obtained by the following sub-query results: $ProdS1(T2, Tom, 2015, TLDKS, L2)$: 0.95, $ConfS2(ODBASE)$: 0.75 and $RankingS4$ $(ODBASE, B)$: 0.65. Note that $AffS3(Tom, L2, 2015)$: 0.85 validates c_{P_1}, but does not interfere in the computation of τ_{out}. □

In our approach, global query answers are restrained by constraints in $\mathcal{C}$. To find an answer t to a query q means to find an instantiation h_t (homomorphism from Definition 2) for the body of q that generates t. Verifying whether the instantiated body of q ($h_t(body(q))$) is valid *w.r.t.* $\mathcal{C}$ ensures the validity of our answer. During this verification process, three situations can be distinguished:
(A) If in $h_t(body(q))$ there is no instantiated atom matching the body of a constraint in $\mathcal{C}$, then the query answer is valid.
(B) If in the set $h_t(body(q))$ there is at least one instantiated atom containing a null value in Δ_N that matches the body of a constraint in $\mathcal{C}$, then the query

answer validity cannot be stated. In this case, the answer has been obtained via an inference rule in Σ (a TGD) on the basis of an unknown value (a fresh null). The presence of this fresh null ensures the existence of an answer, but we do not know whether this answer respects the constraints (since constraints can only be activate by ground atoms). We discard this answer without triggering the constraint verification procedure.
(C) If in the set $h_t(body(q))$ all instantiated atoms that matches the body of a constraint in $\mathcal{C}$ are facts, then the query answer should be tested *w.r.t.* $\mathcal{C}$.

Example 6. In the example of Section 2, consider a query containing $Aff(X_a, X_l, X_y)$ in its body. As $S3$ contains $Prof(Ann, 2015, Tours)$ then the inference produces $Aff(Ann, z_2, 2015)$ (as in Example 1), meaning that Ann is associated to an unknown research laboratory. In the second context, there is a matching with c_{P_2}. As this matching involves a fresh null, this answer is discarded. Figures 2-3 show other instantiations of Aff which activate c_{P_2} in this case. For example, $Aff(Bob, L2, 2015)$ requires $CNRS(L2)$ to be true in $\overline{(\mathcal{S}, \Sigma, \tau)}$. □

The following definition deals with the need of well identifying the set of atoms capable of activating constraints. A constraint cannot be triggered by an atom containing a fresh null (*i.e.*, a piece of unknown information). Indeed, constraint verification cannot always be ensured in the presence of nulls values (which may appear due to TGD).

Definition 6 (Unknown test function). Let I be a set of instantiated atoms. Let $\mathcal{C}$ be a set of constraints. The *unknown test function* states whether a constraint c can be activated on the basis of atoms in set I. Let $unknwon(I, \mathcal{C}) = true$ if there is an atom $\alpha \in body(c)$ for which there exists a homomorphism ν such that $\nu(\alpha) \in I$ and the image of ν contains at least one value $v_{null} \in \Delta_N$. Otherwise $unknwon(I, \mathcal{C}) = false$. □

Now we put together constraint and confidence degree verification to answer global queries. Let $q{:}\tau_{in}$ be a conjunctive global query and $\mathcal{C} = \mathcal{C}_P \cup \mathcal{C}_N \cup \mathcal{C}_K$ be a set of constraints over $\mathbb{G}$. Valid candidate answers are those that respect constraints and are obtained by trusted databases. We do not consider answers for which the unknown test function is true.

Definition 7 (Valid candidate answers). Given a query q restrained by $\mathcal{C}$ with the minimum required truth degree τ_{in}, valid candidate answers of $q{:}\tau_{in}, \mathcal{C}$ over a database $\overline{(\mathcal{S}, \Sigma, \tau)}$ are defined as follows:

$valCandAns(q{:}\tau_{in}, \mathcal{C}, \overline{(\mathcal{S}, \Sigma, \tau)}) = \{(t : \tau_{out}) \mid t$ is a candidate answer as in Definition 5 and h_t is a corresponding homomorphism (Definition 2) and $Unknown(h_t(body(q)), \mathcal{C}) = false$ and $(h_t(body(q))$ is valid *w.r.t.* $(\mathcal{C}_P, \tau_{in})$ and $T^*_{\mathcal{C}_P}(h_t(body(q))$ is valid *w.r.t.* $(\mathcal{C}_N, \tau_{in})$ and $(\mathcal{C}_K, \tau_{in})\}$ □

Definition 7 is completed by the definitions of validity *w.r.t.* each kind of constraints in $\mathcal{C}$.

Definition 8 (Validity *w.r.t.* (C_P, τ_{in})). A set of facts I is valid *w.r.t.* $(\mathcal{C}_P, \tau_{in})$ if for each fact $L \in T^*_{\mathcal{C}_P}(I)$ we have a positive answer for $q() \leftarrow L : \tau_{in}$ on $\overline{(\mathcal{S}, \Sigma, \tau)}$. □

In Definition 7, the computation of the least fixed point of T starts with the literals in the body of the instantiated rule $q(t)$ (where t is a tuple in $ans(q{:}\tau_{in}, \overline{(\mathcal{S}, \Sigma, \tau)})$, as in Definition 5. Thus, the computation of $ans(q{:}\tau_{in}, \overline{(\mathcal{S}, \Sigma, \tau)})$ is the key for obtaining the facts in the initial set I. Then, by finding the least fixed point of $T_{\mathcal{C}_P}$ over I, we produce all the facts which are imposed to be true in $\overline{(\mathcal{S}, \Sigma, \tau)}$ by $\mathcal{C}_P$ and that respect τ_{in}. This is a first filter applied to $ans(q{:}\tau_{in}, \overline{(\mathcal{S}, \Sigma, \tau)})$, since invalid answers are eliminated from the result given to the user. For instance, in Section 2, q_1 instantiation (Figures 2-3) triggers constraint c_{P_1}: *Bob* and *Tom* should be affiliated to $L2$ in year 2015 (a success) while *Joe* should be at $L3$ in 2016 (a failure).

When the body of a negative constraint c matches a fact in a set of facts I, c is triggered to produce $\perp$. Similarly, for key constraints, the triggered constraints will produce an instantiated equality expression. The formalization of this behaviour can be done by applying T, once, on the previously computed set of facts, *i.e.*, we start with $I = T^*_{\mathcal{C}_P}(h_t(body(q))$.

Definition 9 (Validity *w.r.t.* $(\mathcal{C}_N, \tau_{in})$). A set of ground literals I is valid *w.r.t.* $(\mathcal{C}_N, \tau_{in})$ if:

1. $\perp \notin T_{\mathcal{C}_N}(I)$ and
2. $\forall L \in I$ and $c \in \mathcal{C}_N$ of the form $L_1, L_2 \rightarrow \perp$, if there is a homomorphism ν such that[1] $\nu(L_i) = L$, then there is no homomorphism ν' that extends ν and for which $ans(q : \tau_{in}, \overline{(\mathcal{S}, \Sigma, \tau)}) \neq \emptyset$, where $body(q) = \nu'(L_{\bar{i}})$. □

Definition 10 (Validity *w.r.t.* $(\mathcal{C}_K, \tau_{in})$). A set of ground literals I is valid *w.r.t.* $(\mathcal{C}_K, \tau_{in})$ if:

1. All equalities obtained in $T_{\mathcal{C}_K}(I)$ are true (*i.e.*, we do not obtain $a = b$ for two different constants) and
2. $\forall L \in I$ and $c \in \mathcal{C}_K$ of the form $A(\mathbf{Y}, X_1, \mathbf{Z}_1), A(\mathbf{Y}, X_2, \mathbf{Z}_2) \rightarrow X_1 = X_2$ if there is a homomorphism ν such that $\nu(A(\mathbf{Y}, X_1, \mathbf{Z}_1)) = L$, then $ans(q' : \tau_{in}, \overline{(\mathcal{S}, \Sigma, \tau)})$, where q' is the query $q(X_2) \leftarrow \nu(A(\mathbf{Y}, X_2, \mathbf{Z}_2))$, is a singleton containing the tuple value $\nu(X_1)$. □

Definitions 9 and 10 introduce the possibility of generating a new (but simpler) query to certify whether the original query q is valid *w.r.t.* $\mathcal{C}_N$ and $\mathcal{C}_K$.

Example 7. In Section 2, instantiations of $body(q_1)$ contain $Prod(T3, Mary, 2016, ICEIS, L3)$, $Conf(ICEIS)$ and $Ranking(ICEIS, C)$. Constraint c_{P_1} is verified ($Aff(Mary, L3)$ is true) while c_{N_1} and c_{K_1} ($Mary$ is in two labs in 2016) fail. □

[1] In our notation, if $i = 1$ then $\bar{i} = 2$ and vice-versa.

We are now ready to define answers for a constrained query with a required confidence degree. Notice that for identical tuples the result only contains those with higher confidence degree.

Definition 11 (Answers on $\overline{(\mathcal{S}, \Sigma, \tau)}$). Given a global query q, constrained by constraints $\mathcal{C}$ and having the minimum required confidence degree τ_{in}, the set of answers for $(q{:}\tau_{in}, \mathcal{C})$ over $\overline{(\mathcal{S}, \Sigma, \tau)}$ is defined as:

$$ans(q{:}\tau_{in}, \mathcal{C}, \overline{(\mathcal{S}, \Sigma, \tau)})) = \{(t : \tau_{out}) \mid \tau_{out} = max\{\tau^1_{out}, \ldots, \tau^k_{out}\}, \text{ where } (t^1 : \tau^1_{out}) \ldots, (t^k : \tau^k_{out}) \text{ are all in } valCandAns(q{:}\tau_{in}, \mathcal{C}, \overline{(\mathcal{S}, \Sigma, \tau)}) \text{ and } t^1 = \cdots = t^k = t\} \square$$

6 Proof of concept

As a proof of concept we consider our approach over three different query evaluators. These first tests are done over non distributed system and without using confidence degrees. The current implementation of our constraint checking mechanism has two main steps:

1. a first step rewrites a given query q by completing its body according to positive constraints, performs a first $\mathcal{C}_N$ checking and sends q to the evaluator;
2. a second step produces auxiliary simpler queries on the basis of $\mathcal{C}_N$, $\mathcal{C}_K$ and the answers of q obtained in step 1. Optimisation of this second step are under investigation, but for the moment we consider tests where all subsidiary queries are evaluated (and thus in some examples, one such a query may be executed many times). The solution for avoiding invalid pieces of data being checked again and again should be the use of a cache memory.

As a first validator, we use Graal [2], a Java toolkit to query knowledge bases within Datalog$^{\pm}$ (and thus where inference rules can be implemented). Tests were performed on a processor Intel(R) Core(TM) i7, CPU 2.70GHz, 4 Core(s), 8 Logical Processor(s) 16.0GB RAM. For very simple queries, *e.g.* one having just one join and 2 variables, Graal returned 25000 answers in 2s after performing step 1 above over 50000 facts. Our tests validated 7249 answers *w.r.t.* 3 constraints in 100s. For queries requiring more variables and joins (such as q_1 in Section 2) Graal returned 1908 answers over 7250 facts in 8min for step 1. Our tests were faster (2s, 4 constraints, 447 valid answers). In the first example, to verify remaining constraints over initial results, at most 2 sub-queries by answer (*i.e.*, at most 50000 sub-queries) should be evaluated. In the second case, we had at most 5724 sub-queries to test. Moreover, the first database had seven times more facts than the second one.

A second validator uses MySQL (*i.e.*, our validation module connects a MySQL database). We created synthetic tuples on the database schema considered in Section 2 (without the inference rules) and we considered the execution of query q_1 on the second context where all constraints are taken into account. With approximately 2500 tuples we obtained 620 answers after performing step 1

above in $26s$. The second step consisted of validating 1487 queries and was done in $7s$. We obtained 149 answers as our final result. With 15000 tuples the results were: 3477 answers in $1954s$ for step 1; and, a second step performing 9917 validations in $55s$ to obtain 1247 answers.

Our third validator interrogates *DBpedia*[2] data sets after translating our datalog query into a SPARQL query. We have considered an RDF data fragment concerning 5 classes and 5 properties. Our initial query had 8 predicates in its body with a chain join and 6 constraints (2 of each type). Query evaluator here was performed by the Virtuoso SPARQL query service in [1]. We obtained 556 answers after performing step 1 above in $1.9s$ (average of 5 tests). The second step consisted of validating 1878 queries and was done in $77s$ (each query requires a remote connection). We obtained 68 final answers.

The tests with the second and third validators were performed on an Intel(R) Core(TM)2 Duo CPU, $2.53GHz$, 2GiB memory.

Our experiments show that our approach is viable and that we can envisage scalability provided that some optimization tools are implemented (*e.g.* to limit repeating tests). Determining whether it is better to evaluate many simple queries instead of rewriting the user's query into a complex query (completed by constraint requirements, in the lines of step 1 above) is a tricky problem. Efficiency depends on the evaluator, the database schema and instance. For example, we considered MySQL tests without the rewriting phase of step 1: although we had more simple queries to evaluate, we obtained a global evaluation time for q_2, in the second context, of $10s$ instead of $33s$ obtained when applying the step 1 above. Thus a finer analysis of database schema and instance may be needed to efficiently choose one of the following implementation strategy: query rewritten to take constraints into account or evaluating many simple queries.

7 Group-by queries

Towards our goal of collecting data for further analysis, we summarize our language extension (inspired in [19]) to allow a *group by query* $q_{ag} : \tau_{in}$ which is a formula of the form:

$$q(X_{i_1}, \ldots, X_{i_k}, a_1, \ldots, a_m) \leftarrow aggr(\phi(X_1, \ldots, X_n), i_1, \ldots, i_k, \\ a_l = func_l(X_{j_1}, \ldots, X_{j_{k_l}})) \text{ for each } 1 \leq l \leq m \\ \text{with a required confidence degree } \tau_{in} \text{ where:}$$

aggr is a second order predicate; $\phi(X_1, \ldots, X_n)$ is a conjunction of atoms over a schema; $i_1, \ldots, i_k$ are positions in $[1, n]$; $X_{i_1}, \ldots, X_{i_k}$, and $X_{j_1}, \ldots, X_{j_{k_l}}$ are variables in $\{X_1, \ldots, X_n\}$; $a_1 \ldots a_m$ are new variables not existing in $\{X_1, \ldots, X_n\}$; $func_l$ is a function which returns a real value.

The answer of $(q_{ag} : \tau_{in}, \mathcal{C})$ over $\overline{(\mathcal{S}, \Sigma, \tau)}$, denoted by $ans(q_{ag} : \tau_{in}, \mathcal{C}, \overline{(\mathcal{S}, \Sigma, \tau)})$ is the set of $t : \tau_{out}$ such that each tuple t is the concatenation of a tuple u and m real values a, *i.e.*, $t = u.a_1. \ldots .a_m$ (for $m \geq 0$) defined as follows:

[2] http://mappings.dbpedia.org/server/ontology/classes/

- $u : \tau_{out} \in ans(q_1(X_{i_1}, \ldots, X_{i_k}) : \tau_{in}, \mathcal{C}, \overline{(\mathcal{S}, \Sigma, \tau)})$ and h_u is its associated homomorphism, *i.e.*, one used to find u as an answer to the conjunctive query $q_1(X_{i_1}, \ldots, X_{i_k}) \leftarrow \phi(X_1, \ldots, X_n)$ with the minimal expected confidence degree τ_{in}.
- For each real value a, we have $a = func(ans(q_2 : \tau_{in}, \mathcal{C}, \overline{(\mathcal{S}, \Sigma, \tau)})$ where q_2 is the conjunctive query $q_2(X_{j_1}, \ldots, X_{j_{k_l}}) \leftarrow h_u(\phi(X_1, \ldots, X_n))$. The body of q_2 is an instantiated version of the body of q_1 and that instantiation is done with h_u (*i.e.*, $h_u(\phi(X_1, \ldots, X_n)) = h_u(body(q_1))$). The answer of q_2 is composed by k-tuples. Each a value is obtained by the application of a function on this set of tuples.

Example 8. Consider query $q(X_l, X_y, X_r, a) \leftarrow aggr(Prod(X_t, X_a, X_y, X_@, X_l), Conf(X_@), Ranking(X_@, X_r), X_l, X_y, X_r, a = count(X_t))$ with $\tau_{in} = 0.60$ on the first context mentioned in Section 2. Here, to simplify, we abuse of the notation using X_l, X_y, X_r to also indicate positions. The answer over instance in Figures 2 and 3 is $(L2, 2015, B, 2) : 0.65$ since authors Bob and Tom from $L2$ published in $TLDKS$ in 2015. □

8 Related work and further discussion

In this paper, we propose a user-friendly environment to query a distributed semantic data graph (such as RDF documents) that ensures reliable results even when data source quality is not guaranteed. Our query environment would evolve to deal with a big amount of distributed data. Our long-term goal is to produce a user-friendly tool for helping graph analysis.

Queries, constraints and confidence. Ontological queries (such as in [6,7,9,12]) have inspired our query language on graph databases. Contrary to them, our positive constraints are not just inferences but are closer to traditional database constraints (*e.g.* if $A(a)$ is imposed as true by a constraint c_P, sources should have this information; c_P does not infer it). Our constraints are triggered by instantiated atoms in the query body (a step towards efficient treatment of large amount of data). Only key constraints are treated in [17] and an answer is given if it is restricted to query positions not constrained by a key. For instance, the query $q(Z) \leftarrow r(X, Y, Z)$ in the presence of a key constraint $r(X, Y_1, Z_1), r(X, Y_2, Z_2) \rightarrow Y_1 = Y_2$ and data $r(a, b, c)$, $r(a, b', c)$ offers c as an answer. This result is discarded in our approach due to constraint violation. Indeed, for us, answers are issued from facts (in the query body) taking no part in a constraint violation. The proposal in [15] is close to the one in [17] but, in addition, similarly to us, provenance information is used in answer computation. In [18] we find different semantics for query answering over inconsistent Datalog$^{\pm}$ ontologies. Their goal is to propose corrections to the database, while ours is to avoid answering on the basis of inconsistent data. Different work deals with unwilling data (*e.g.* , [4,5,11,23]). An important characteristic in these approaches is that an answer may be considered true even if it is obtained with confidence degree

inferior to 1; avoiding the strictness point of view of classical logic that eliminates results not responding entirely to constraints. Our approach is inspired in [4,5] where a fuzzy datalog is presented. Our originality is to allow parametrized use of confidence information. We can also envisage extensions where a minimal confidence degree is associated to constraints and where result is sorted according to confidence degree – augmenting the flexibility of our approach.

Implementation aspects. Our proposal allows the implementation of an independent module capable of interacting with different query evaluation mechanisms. A global query is sent to the evaluator and answers are filtered according to the established context (fixed by the global constraints). In this paper, we have described the whole system in a general manner; imagining sources also expressed in a logic programming formalism, with inferences. However, one can envisage different source formats.

For instance, our module can interact with distributed relational sources. A global query can be rewritten according to a LAV method such as Minicon [21] and use MapReduce-based solutions – some MapReduce solutions have already been proposed for dealing with RDF management at large scale (see [16] for a survey). To deal with graph databases (and, thus, RDF data) multiple MapReduce jobs are required in order to evaluate a given query. Optimization in this domain focus on the reduction of the number of MapReduce jobs ([14]). Thus, our query evaluation would be done by steps according to the required joins.

Our module can also interact with distributed systems such as the one used in [20] where distributed fragments group data according to allocation patterns. In this context, our query can be processed by exploring the existence of these patterns. Each server executes in parallel the same query plan, using asynchronous BSP (Bulk Synchronous Parallel) computation model. A server s computes, locally, sub-query answers depending on locally stored data. For those sub-queries whose data are not available locally, s either requests the remote serves to transmit data or yields to them the rest of the computation. When slaves finish computing the query plan, their results are sent to the master.

We are currently concerned by our approach optimization and employment over large amount of RDF data. In this context, we intend to explore two directions. The first one consists in deploying additional frameworks running over Hadoop, and, to this end, we should propose an efficient translator of our datalog query into Pig Latin or Hive. The second one consists in using the system proposed in [20] as an evaluator of our query.

Given a query q, it is worth noting that our approach may require multiple auxiliary query in order to answer q *w.r.t.* $\mathcal{C}$. As already said, our current implementation rewrites q *w.r.t.* $\mathcal{C}_P$, before sending it to the evaluator. Negative and key constraints could be treated similarly if the evaluator is capable of (efficiently) evaluating negation (currently, this is not the case, for instance, for [20]). Notice however that rewriting is an implementation option: an automatic way for imposing our constrained environment. The other option, is the production of auxiliary simple queries (mostly boolean queries). This option can be optimized

by the use of cache memories. We should therefore evaluate how to implement efficiently our approach in accordance with the (lower-level) evaluator.

9 Conclusions and perspectives

Our querying semantics originality is summarized by the following aspects:
(1) A user can settle a *personalised* context where query answers are filtered. Constraints, imposed on queries and not on data sources, are an important factor of this personalisation. No correction trial is performed on an inconsistent database but answers are ensured to be valid *w.r.t.* the established context.
(2) Data confidence according to provenance is taken into account. The user can parametrize how information concerning source confidence is taken into account. Even if in this paper, examples given use a very simple strategy, our semantics allows more sophisticated approaches.
(3) Our approach can be implemented in an independent module, offering flexibility to the user. Freedom benefits user in two different levels: (A) Different choices are possible for the lower level query evaluation mechanism; demanding only relatively simple interfaces. (B) Context parameters (including constraints and confidence degrees) may be settled for a group of queries, but changed for another group of queries.
(4) The use of a querying constrained environment is a lighter and customized tool that facilitates data access under given quality requirements. Sophisticated queries might integrate restrictions and, in this way, simulate the settled context. However, the use of a set of simple queries to impose constraints may remedy the lack of lower level mechanisms capable of evaluating non conjunctive queries.

Our overall goal is to build a parametrized environment to produce valid and reliable results even when the source data quality cannot be entirely ensured – an important aspect when dealing with large amount of data. To achieve this goal, we are currently working on implementation and optimization devices such as the use of caches to avoid repeating validity checks. As a future research direction we consider language extensions to deal with complex objects as needed in later data analysis.

References

1. Dbpedia endpoint. At http://dbpedia.org/sparql.
2. Graal. At https://graphik-team.github.io/graal/.
3. S. Abiteboul, I. Manolescu, P. Rigaux, M.-C. Rousset, and P. Senellart. *Web data management*. Cambridge University Press, 2011.
4. Á. Achs. Computed answer from uncertain knowledge: A model for handling uncertain information. *Computers and Artificial Intelligence*, 26(1):63–76, 2007.
5. Á. Achs and A. Kiss. Fuzzy extension of datalog. *Acta Cybern.*, 12(2):153–166, 1995.
6. A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.

7. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
8. W. Fan and J. Huai. Querying big data: Bridging theory and practice. *J. Comput. Sci. Technol.*, 29(5):849–869, 2014.
9. F. Goasdoué, V. Lattès, and M. Rousset. The use of CARIN language and algorithms for information integration: The PICSEL system. *Int. J. Cooperative Inf. Syst.*, 9(4):383–401, 2000.
10. L. Gomes Jr., B. Amann, and A. Santanchè. Beta-algebra: Towards a relational algebra for graph analysis. In *Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference (EDBT/ICDT), Brussels, Belgium, March 27th*, pages 157–162, 2015.
11. G. Gottlob, T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Query answering under probabilistic uncertainty in datalog+ / - ontologies. *Ann. Math. Artif. Intell.*, 69(1):37–72, 2013.
12. G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 2–13, 2011.
13. M. Halfeld Ferrari Alves, D. Laurent, and N. Spyratos. Update rules in datalog programs. *J. Log. Comput.*, 8(6):745–775, 1998.
14. M. F. Husain, P. Doshi, L. Khan, and B. M. Thuraisingham. Storage and retrieval of large RDF graph using hadoop and mapreduce. In *Cloud Computing, First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings*, pages 680–686, 2009.
15. E. J. Kao. Consistency and provenance in rule processing. In *Rule-Based Modeling and Computing on the Semantic Web, 5th International Symposium, RuleML 2011-America, Ft. Lauderdale, FL, Florida, USA, November 3-5, 2011. Proceedings*, pages 66–80, 2011.
16. Z. Kaoudi and I. Manolescu. RDF in the clouds: a survey. *VLDB J.*, 24(1):67–91, 2015.
17. P. G. Kolaitis, E. Pema, and W. Tan. Efficient querying of inconsistent databases with binary integer programming. *PVLDB*, 6(6):397–408, 2013.
18. T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Inconsistency handling in datalog+/- ontologies. In *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, pages 558–563, 2012.
19. I. S. Mumick, H. Pirahesh, and R. Ramakrishnan. The magic of duplicates and aggregates. In *Proceedings of the 16th International Conference on Very Large Data Bases*, VLDB '90, pages 264–277. Morgan Kaufmann Publishers Inc., 1990.
20. R. Penteado, R. Schroeder, and C. S. Hara. Exploring controlled RDF distribution. Submitted.
21. R. Pottinger and A. Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB J.*, 10(2-3):182–198, 2001.
22. A. Shkapsky, M. Yang, M. Interlandi, H. Chiu, T. Condie, and C. Zaniolo. Big data analytics with datalog queries on Spark. To appear in SIGMOD 2016.
23. G. Stoilos, N. Simou, G. B. Stamou, and S. D. Kollias. Uncertainty and the semantic web. *IEEE Intelligent Systems*, 21(5):84–87, 2006.