Categorize By: Deductive Aggregation of Semantic Web Query Results

Claudia d'Amato[1], Nicola Fanizzi[1], and Agnieszka Ławrynowicz[2]

[1] Dipartimento di Informatica, Universita degli Studi di Bari, Italy
{claudia.damato,fanizzi}@di.uniba.it,
[2] Institute of Computing Science, Poznan University of Technology, Poland
alawrynowicz@cs.put.poznan.pl

Abstract. Query answering on a wide and heterogeneous environment such as the Web can return a large number of results that can be hardly manageable by users/agents. The adoption of grouping criteria of the results could be of great help. Up to date, most of the proposed methods for aggregating results on the (Semantic) Web are mainly grounded on syntactic approaches. However, they could not be of significant help, when the values instantiating a grouping criterion are all equal (thus creating a unique group) or are almost all different (thus creating one group for each answer). We propose a novel approach that is able to overcome such drawbacks: given a query in the form of a conjunctive query, grouping is grounded on the exploitation of the semantics of background ontologies during the aggregation of query results. Specifically, we propose a solution where answers are deductively grouped taking into account the subsumption hierarchy of the underlying knowledge base. As such, the results can be showed and navigated similarly to a faceted search. An experimental evaluation of the proposed method is also reported.

1 Introduction

The users of the (Semantic) Web often perform interactive, and exploratory data retrieval, where queries often result in an overwhelming number of the returned answers. However typically, only a small part of the result set is relevant to the user thus making necessary the analysis of the retrieved results to identify those relevant ones. This phenomenon, known as *information overload*, constitutes a ceaseless challenge for researchers. It may occur in situations when the user, at the beginning of a search, submits a broad query to avoid exclusion of possibly interesting results, only further possibly reformulating it into a more precise narrower query. It is also inherent to the task of browsing through a collection of resources instead of searching among them. Manually separating the interesting items from the uninteresting ones is a tedious and time consuming job. For this reason, various services have been set up to manage the information overload. They may be categorized as variations of two complementary approaches: (a) predefined category structures; (b) fully automatic search engines.

Predefined category structures are maintained by humans, who organize resources into a hierarchy, grouping appropriate ones together under a meaningful

category description. This is meant to support user browsing activity. The main drawback of such an approach is that the categories are frequently obsolete, since they are not updated so quickly as new documents and new topics appear. In turn, the mechanisms used by automatic search engines, enable them to stay relatively up-to-date, what makes the search engines the premier choice for most Web users. However, when the number of the results is huge, even despite their ranked order, manually investigating them is a big effort without any additional navigation tools. In order to facilitate browsing and managing results of a Web search/query, methods for grouping answers on the ground of user defined criteria would be exploited.

In this paper, we propose to marry the benefits of two complementary approaches for handling information overload by offering a method that, given a query in the form of a conjunctive query [1–5], produces a dynamic categorization over ranked query results. The key feature of the method is in the exploitation of the *semantics* of knowledge bases of reference (in the form of ontologies) for grouping results with respect to a user defined criterion. Specifically, given a certain grouping criterion, expressed as a (complex) concept from a knowledge base of reference, results are grouped in agreement with (part of) the subsumption hierarchy deductively obtained by considering the specified concept and the given ontology.

Currently, besides of the usual syntactic based approach (grounded on the use of keywords) adopted by the main Web search engines such as Google or Yahoo, the other approaches that are generally exploited for grouping structured query answers are based on the semantics drawn from SQL, the standard language for querying relational databases. An example is given by the implementation of the aggregation operators (COUNT, MIN, MAX, AVG, SUM, GROUP BY) for SPARQL in Virtuoso[3]. However, besides of the fact that these implementations are not part of the SPARQL standard, these latter approaches (differently from the method that we propose) do not assume to perform any reasoning involving background knowledge during the query results aggregation, and as such they are purely syntactic as well. The same happens for grouping/aggregating features that are to be included in the ongoing SPARQL 2 standard.

The main contributions of the paper are summarized as follows: (a) we introduce a novel way for grouping query answers (*semantic grouping*) where grouping is done on the ground of a knowledge base of reference; (b) we propose a technique for aggregating query results consisting on a dynamic generation of a navigable hierarchy on top of the retrieved results and that is based on their semantics. Such a hierarchy constitutes a multi-valued classification of the results that may be seen as a novel approach for generating a *dynamic faceted classification* over retrieved results, enabling further faceted search/browsing over them; (c) we present the experimental results of the application of our proposed method to ontologies expressed in *Web Ontology Language (OWL)*[4], whose semantics is based on *description logic (DL)* [6].

[3] http://virtuoso.openlinksw.com/
[4] http://www.w3.org/TR/owl-features/

The rest of the paper is organized as follows. In Sect. 2 a motivating example for our work is presented. In Sect. 3 we introduce the basics of the knowledge representation formalism of choice: description logics and the notion of conjunctive queries over DL knowledge bases. In Sect. 4 we present our proposed approach. Sect. 5 reports on the experimental evaluation of our method while Sect. 6 discusses the work related to ours. In Sect. 7, conclusions are drawn.

2 Motivating example

Let us discuss the need for and advantage of semantic query answer aggregation on the ground of a motivating example.

Example 1 (Motivating scenario). Maria and Sebastian are searching for a weekend break offer. In order to get faster insight to the retrieved results they would like to have the answers grouped by a destination criterion. After getting first insight to the results, they perform further exploration, and hence they submit another query, most specific one, asking for destinations located in mountains, and for budget accommodations offered in these destinations with a request to group results w.r.t. destinations, and additionally w.r.t. accommodations offered.

Let us suppose that the Web service mentioned in the example exploits a SPARQL endpoint that retrieves the results from Semantic Web datasets. Furthermore the Web service uses an ontology as background knowledge on the given domain (an instance is presented in Example 3). In the following, instances of SPARQL queries for the needs illustrated in Example 1 are showed.

Example 2 (Example queries in SPARQL syntax).

$Q_{1\_group\_by}$ = `SELECT ?x ?y WHERE { ?x rdf:type :WeekendBreakOffer . ?x :hasDestination ?y } GROUP BY ?y`

$Q_{2\_group\_by}$ = `SELECT ?x ?y ?z WHERE { ?x rdf:type :WeekendBreakOffer. ?x :hasDestination ?y . ?y :locatedIn ?v . ?v rdf:type :Mountains . ?x :hasAccomodation ?z . ?z rdf:type :BudgetAccomodation } GROUP BY ?y ?z`

Let us assume an application of the classical semantics of the `GROUP BY` clause known from SQL in the evaluation of the query Q_1. Let us assume further that instances that bind to variable y are town names. In such a case, the results will be partitioned so that one row for each town name is created. Considering that there may be possibly many towns satisfying the query conditions, there will be also too many groups to provide significant added value.

Note also that classical semantics of `GROUP BY` clause, which is to partition the results by identical values, disregards the presence of any background knowledge, even if there is available some. Consider for example that the towns are annotated by terms from the ontology such as City, EuropeanDestination, ItalianDestination, PolishDestination. Considering such annotations as a grouping condition, it would be possible to aggregate the results into a smaller number of 'semantic' groups. The classical semantics of `GROUP BY` disregards also semantic

relationships between groups like subclass-superclass relation (e.g. ItalianDestination, and EuropeanDestination). Exploiting the background ontology would enable making such relations explicit.

Another problem with the syntactic approach may happen when all the results in the grouping condition, e.g. all destinations in our case, refer to the same town. This gives as grouping result a unique row which synthesizes all instances, thus giving almost null information.

Summarizing, a merely syntactic approach, as the one used by the `GROUP BY` clause in the database context, could not be of great help when several destinations are found, or when the results refer to the same destination. To manage situations like these, a *semantic group by* could be adopted, namely a method that is able to group query results on the ground of a knowledge base of reference.

3 Preliminaries

3.1 Representation and Inference

Description logics (*DLs*) [6] are a family of knowledge representation languages (endowed with a model-theoretic semantics and reasoning services) that have been adopted as theoretical foundation for OWL language [7]. Basic elements in DLs are: *atomic concepts* (denoted by A) and *atomic roles* (denoted by R, S). *Complex descriptions* (denoted by C and D) are inductively built by using concept and role *constructors*.

Semantics is defined by *interpretations* $\mathcal{I}=(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where non-empty set $\Delta^{\mathcal{I}}$ is the domain of the interpretation and $\cdot^{\mathcal{I}}$ is an interpretation function which assigns to every atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and to every atomic role R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended to complex concept descriptions by the inductive definition as presented in Tab. 1. A DL *knowledge base*, KB, is formally defined as: $\mathcal{KB} = (\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ is called

Table 1: Syntax and semantics of example DL constructors.

Constructor	Syntax	Semantics
Universal concept	$\top$	$\Delta^{\mathcal{I}}$
Bottom concept	$\bot$	$\emptyset$
Negation of arbitrary concepts	$(\neg C)$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
Intersection	$(C \sqcap D)$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Union	$(C \sqcup D)$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Value restriction	$(\forall R.C)$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$
Full existential quantification	$(\exists R.C)$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$

a TBox and it contains axioms defining concepts, and $\mathcal{A}$ is called an ABox and it contains assertions about individuals such as $C(a)$ (that means that the invidual a is an instance of the concept C) and $R(a,b)$ (that means that a is R-related to b).

As regards the inference services, we recall the definition of *instance-checking*

and *subsumption* that are further used in the paper. *Instance-checking* amounts to determine whether an individual, say a, belongs to a concept extension, i.e. whether $C(a)$ holds for a certain concept C. As regards *subsumption*, given two concept descriptions C and D in a TBox $\mathcal{T}$, C subsumes D (denoted by $D \sqsubseteq C$) if and only if, for every interpretation $\mathcal{I}$ of $\mathcal{T}$ it holds that $D^{\mathcal{I}} \subseteq C^{\mathcal{I}}$. C equivalent to D (denoted by $C \equiv D$) amounts to $C \sqsubseteq D$ and $D \sqsubseteq C$.

3.2 Conjunctive Queries

Queries admitted in this work are expressed in the form of *conjunctive queries* [1–5]. Let N_C, N_R, N_I be the sets of *concept names*, *role names* and *individual names* respectively and let N_V be a countably infinite set of variables disjoint from N_C, N_R, and N_I. Let by $\mathbf{x}$ and $\mathbf{y}$ denote the sets of distinguished and nondistinguished variables, respectively where $\mathbf{x}, \mathbf{y} \subseteq N_V$. A conjunctive query, denoted with $Q(\mathbf{x}, \mathbf{y})$, is a finite conjunction of a non-empty set of *atoms*. An *atom* is an expression of kind $A(t_1)$ (concept atom) or $R(t_1, t_2)$ (role atom), where A is a concept name, R is a role name, and t_1 and t_2 are individuals from N_I or variables from $\mathbf{x}$ or $\mathbf{y}$. $Var(Q)$ ($Var(at)$) denote the set of variables occurring in the query Q (atom at).

An answer to a query $Q(\mathbf{x}, \mathbf{y})$ w.r.t. KB is an assignment θ of individuals to distinguished variables such that $KB \models \exists \mathbf{y} : Q(\mathbf{x}\theta, \mathbf{y})$.

Example 3 (Conj. Query). Given the following knowledge base:
$\mathcal{T}$ = { City $\sqsubseteq$ Destination, EuropeanDestination $\sqsubseteq$ Destination, ItalianDestination $\sqsubseteq$ EuropeanDestination, PolishDestination $\sqsubseteq$ EuropeanDestination, Hotel $\sqsubseteq$ Accomodation, BudgetAccomodation $\sqsubseteq$ Accomodation, B&B $\sqsubseteq$ BudgetAccomodation, Hostel $\sqsubseteq$ BudgetAccomodation, SkiingSite $\sqsubseteq$ Site, SightSeeingSite $\sqsubseteq$ Site, $\top \sqsubseteq \forall$ hasSite.Site, $\top \sqsubseteq \forall$ hasDestination.Destination, $\top \sqsubseteq \forall$ hasAccomodation.Accomodation}.
$\mathcal{A}$ = { locatedIn(ZAKOPANE, TATRA),hasSite(ZAKOPANE, SKI_LIFTS_NOSAL), SkiingSite(SKI_LIFTS_NOSAL), locatedIn(CHOCHOLOW, TATRA), hasSite(CHOCHOLOW, HIGHLANDERS_WOODEN_HOUSES), SightSeeingSite(HIGHLANDERS_WOODEN_HOUSES), Mountains(TATRA), WeekendBreakOffer(O1), hasAccomodation(O1,A1), B&B(A1), hasDestination(O1, ZAKOPANE),PolishDestination(ZAKOPANE), WeekendBreakOffer(O2),hasAccomodation(O2,A2),B&B(A2), hasDestination(O2, CHOCHOLOW), PolishDestination(CHOCHOLOW), WeekendBreakOffer(O3), hasDestination(O3, ROME), City(ROME), WeekendBreakOffer(O4), hasDestination(O4, PISA), ItalianDestination(PISA)}.

we formalize conjunctive queries corresponding to the ones from Example 2 as:

$$Q_1(x, y) = \mathsf{WeekendBreakOffer}(x) \wedge \mathsf{hasDestination}(x, y)$$

$$Q_2(x, y, z) = \mathsf{WeekendBreakOffer}(x) \wedge \mathsf{hasDestination}(x, y) \wedge \mathsf{locatedIn}(y, v) \wedge \mathsf{Mountains}(v) \wedge \mathsf{hasAccomodation}(x, z) \wedge \mathsf{BudgetAccomodation}(z)$$

4 Deductive Aggregation: Semantic Grouping

Let us consider the knowledge base and the example queries from Example 3. Supposing the case in which many offers are available, the user would be interested in grouping the results with respect to the different destinations. However, as already discussed in the previous section, a merely syntactic approach, as the one used by the *group by* clause in the database context, could not be of great help in cases in which several destinations are found or all results refer to the same destination. In order to manage cases like this, a *semantic group by* could be adopted, namely a method that is able to group query results on the ground of a knowledge base of reference. Specifically, looking at the knowledge base in the example above, results could be grouped on the ground of the pertaining country of a destination (ItalianDestination, PolishDestination).

In this paper we present a method that is able to perform the *semantic group by* on the ground of concepts that are more specific (in the knowledge base of reference) with respect to the concept that is adopted by the user for grouping the results. The method is presented in the following parts of this section.

4.1 Basics of the Semantic Aggregation

The general idea behind *semantic group by* is to categorize the results with regard to concept hierarchies inferred for each variable appearing in a grouping condition. Thus, in order to formalize our proposition of a *semantic group by*, we introduce a special second order predicate categorize_by as presented in the following definition.

Definition 1 (categorize_by). *A conjunctive query with a* semantic aggregate subgoal *is of the form*

$$categorize\_by([X_1, X_2, ..., X_m]) : Q(\mathbf{x}, \mathbf{y})$$

where $[X_1, X_2, ..., X_m]$ *is a grouping list of variables appearing in* $\mathbf{x}$.

In Example 4 we present the conjunctive queries with the categorize_by clause for the scenario discussed throughout this paper.

Example 4 (Example queries with the categorize_by clause).

$Q_{1\_categorize\_by}(x, y) = categorize\_by(y) : \mathsf{WeekendBreakOffer}(x) \wedge \mathsf{hasDestination}(x, y)$

$Q_{2\_categorize\_by}(x, y, z) = categorize\_by(y, z) : \mathsf{WeekendBreakOffer}(x) \wedge \mathsf{hasDestination}(x, y)$
$\wedge \mathsf{locatedIn}(y, v) \wedge \mathsf{Mountains}(v) \wedge \mathsf{hasAccomodation}(y, z) \wedge \mathsf{BudgetAccomodation}(z)$

Further, we formally define the notion of *semantic category*.

Definition 2 (Semantic category). *Given is query*

$$Q = categorize\_by([X_1, X_2, ..., X_m]) : Q(\mathbf{x}, \mathbf{y})$$

A semantic category *is a tuple of concepts* $\langle C_1, C_2, \ldots, C_m \rangle$*, where each* C_i *corresponds to* X_i *in the grouping variables list of* Q.

Semantic categories form a partially ordered multi-valued classification $\mathcal{H}$ induced by a subsumption relation between concepts appearing in the same place in tuples. Then, the operational semantics for categorize_by clause is to first create $\mathcal{H}$, a partially ordered set of semantic categories, based on inferred semantic types of grouping variables, and then to partition the input relation to groups with equal values for the same semantic categories.

4.2 Inferring the Type of the Variables

Let v_i be a query variable used in the grouping clause of a query. For each such variable a concept is derived. Firstly, the explicit typing represented by those concepts explicitly mentioned in the query atoms $C(v_i)$ is considered. Additionally, the implicit types inferred by the role atoms $R(v_i, \cdot)$ or $R(\cdot, v_i)$, respectively the domain and range of role R are added. Now, let $\mathcal{B}_i := \{C_1^i, \ldots, C_{n_i}^i\}$ be the set of concepts describing individuals the variable v_i can be bound to according to the query atoms. The final concept determined for each query variable is:

$$C^i := \bigsqcap_{C_p^i \in \mathcal{B}} C_p^i$$

A concept corresponding to each query variable included in the `CATEGORIZE BY` clause is derived. Then it is classified in the subsumption hierarchy of the concepts in the knowledge base. A sub-hierarchy of concepts rooted at these concepts will be used in the next step, which determines the final multivalued classification of semantic categories used for grouping the query answers.

Example 5 (Determining the type of variable). Consider query $Q_{2\_categorize\_by}(x, y, z)$ from Example 4, and variable z appearing in the grouping variables list. Then the concept inferred for the variable z is:

$$C^z := \mathsf{Accomodation} \sqcap \mathsf{BudgetAccomodation}$$

4.3 Constructing a Tree of Semantic Categories

The primary motivation for this work is to enable better results exploration for the user by reducing information overload by means of presenting him/her the groupings of the results. From this perspective we are rather interested in some compromise between generating a complete multivalued classification of semantic categories versus meeting the demands of real time computation of the results, and improving the user experience, not overloading him/her with too many groupings (the initial problem). In particular we propose to concentrate on subtrees of concepts, generated for each grouping variable, and on a tree product as a mean to obtain a tree-shaped classification of semantic categories. In general, however, one may adopt different solutions, what is also discussed in this section.

Concept Hierarchy for a Single Variable A basic approach for generating a sub-hierarchy of concepts to be rooted at the concept C^i, corresponding to a grouping variable v_i, is to reproduce a part of a classified subsumption hierarchy produced by a DL reasoner.

In some cases, however, this may not be enough to obtain meaningful hierarchy of groups. Consider for example the situation, where the concept C^i is a leaf in the classified subsumption hierarchy, e.g. PolishDestination in the knowledge base from Example 3. Another case where this basic approach would not be of help is when all the retrieved results for variable v_i, e.g. all destinations from the discussed example, fall under the same type, e.g. again PolishDestination. This would happen in case of query $Q_{2\_categorize\_by}(x, y, z)$ from Example 4.

In such cases more advanced approach could be adopted, that is based on an iterative application of a concept refinement operator, for example adapted from such concept specialization operators like the ones proposed in [8–10] to the case of purely deductive approach. The application of the refinement operator would introduce further, unnamed, complex concepts into the sub-hierarchy rooted at C^i. For example, the concept PolishDestination could be then specialized into the concepts: PolishDestination $\sqcap$ $\exists$hasSite.SkiingSite and PolishDestination $\sqcap$ $\exists$hasSite.SightSeeingSite to differentiate two Polish destinations present in the example knowledge base, ZAKOPANE, and CHOCHOLOW, based on the sites they offer.

Multiple Variables: Tree Product Let us define the subtrees trees by the quadruple (N, E, r, ℓ), where N is the set of nodes, E is the set of edges between nodes (n_a, n_b), $r \in N$ stands for the root node and ℓ is the labeling function assigning each node with the corresponding concept.

Now, let us consider the case of two variables, say v_i and v_j in the `CATEGORIZE BY` clause and their related subtrees, $T_i = (N_i, E_i, r_i, \ell_i)$ and $T_j = (N_j, E_j, r_j, \ell_j)$, in the subsumption hierarchy. Multiple variables will be processed by iterating the binary product operation presented in the following. A formal sketch of the algorithm is shown in Figure 1. The product $T_i \times T_j := (N_{ij}, E_{ij}, r_{ij}, \ell_{ij})$ of the two subtrees is computed recursively. The root of the product tree is made up by the node $r_{ij} = \langle r_i, r_j \rangle \in N_{ij}$ labeled with $\ell_{ij}(r_{ij}) = \langle \ell(r_i), \ell(r_j) \rangle$. For each couple of children nodes c_i of r_i and c_j of r_j, a new child $r' = \langle c_i, c_j \rangle$ of r_{ij} is obtained. Hence r' is the root of the subtree that constitutes the product T' of the subtrees T'_i and T'_j, rooted respectively, at c_i and c_i. The product subtree T' is finally connected to the product tree under construction T_{ij}.

An important feature of the product is that its complexity is polynomial (note that if the trees are represented with matrices then the proposed product corresponds to a matrix product).

Figure 2 illustrates the general idea of generation of the tree product, and Figure 3 illustrates the tree product by means of an example following our motivating scenario.

Alternative Tree Products In some cases the lever-wise computation of product trees would yield poorly structured trees, when one of them is rooted with

$\times(T_i, T_j) : (N_{ij}, E_{ij}, r_{ij}, \ell_{ij})$
begin
$E_{ij} \leftarrow \emptyset$
$r_{ij} \leftarrow \langle r_i, r_j \rangle$
$N_{ij} \leftarrow \{r_{ij}\}$
$\ell_{ij}(r_{ij}) \leftarrow \langle \ell(r_i), \ell(r_j) \rangle$
for each (c_i, c_j) **where** $(r_i, c_i) \in E_i$ **and** $(r_j, c_j) \in E_j$ **do**
    $T_i' \leftarrow \text{SUBTREE}(T_i, c_i)$
    $T_j' \leftarrow \text{SUBTREE}(T_j, c_j)$
    $(N', E', r', \ell') \leftarrow \times(T_i', T_j')$
    $N_{ij} \leftarrow N_{ij} \cup N'$
    for each $node \in N'$ **do**
        $\ell(node) \leftarrow \ell'(node)$
    $E_{ij} \leftarrow E_{ij} \cup E' \cup \{\langle r_{ij}, r' \rangle\}$
return $(N_{ij}, E_{ij}, r_{ij}, \ell_{ij})$
end

Fig. 1: Tree product operator.

a primitive concept, for example. In such cases it may be advisable to use alternative definition of the tree product operator (derived from graph product operators), which is able to retain all of the structural information contained in the subsumption hierarchies.

Definition 3 (alternative tree products). *Given two trees* $T_1 = (V_1, E_1, r_1, \ell_1)$ *and* $T_2 = (V_2, E_2, r_2, \ell_2)$*, their* product*, denoted* $T_1 \times T_2$*, is a tree* $T_p = (V_p, E_p, r_p, \ell_p)$ *defined as follows:*

- *the vertex set is a subset of the Cartesian product* $V_p \subseteq V_1 \times V_2$*;*
- $r_p = (r_1, r_2) \in V_p$*;*
- $\ell_p(u, v) = (\ell_1(u), \ell_2(v))$*;*
- *two product vertices* $(u_1, u_2) \in V_p$ *and* $(v_1, v_2) \in V_p$ *are connected in* T_p*, i.e.* $((u_1, u_2), (v_1, v_2)) \in E_p$*, iff* u_1, u_2, v_1, v_2 *satisfy the conditions specified in the following table:*

Co-normal product	$(u_1, v_1) \in E_1 \vee (u_2, v_2) \in E_2$
Tensor product	$(u_1, v_1) \in E_1 \wedge (u_2, v_2) \in E_2$
Rooted product	$\bigcup_{u \in V_1} \{((u, v), (u, v')) \mid (v, v') \in E_2\}$

4.4 From Trees to Groups

Having produced a tree of semantic categories, the next step is to assign results (tuples) to the semantic categories. We propose to perform it incrementally in line with the ranking of the results provided by the query answering engine. For each tuple only the projection of the bindings of variables has to be taken into account (that is only bindings of the variables appearing in the `CATEGORIZE BY` clause). Then starting from the root of a tree and following paths to its

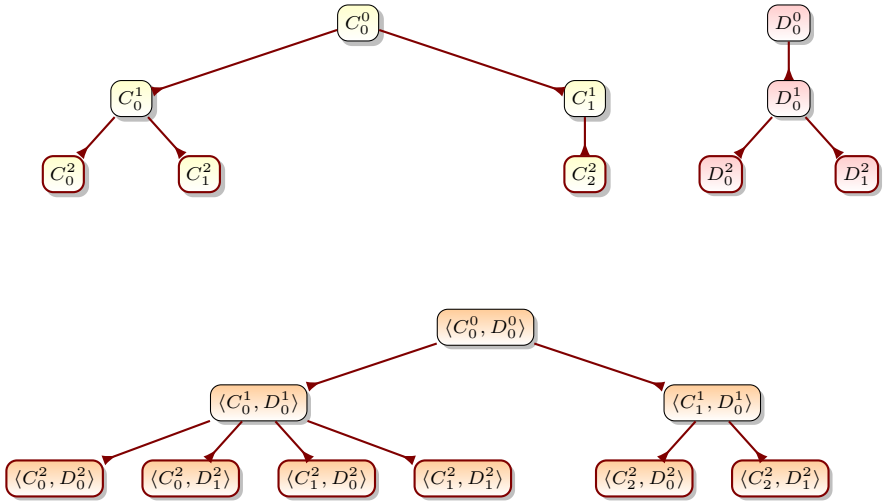


Fig. 2: Tree product example: the lower tree is the product of the two upper trees.

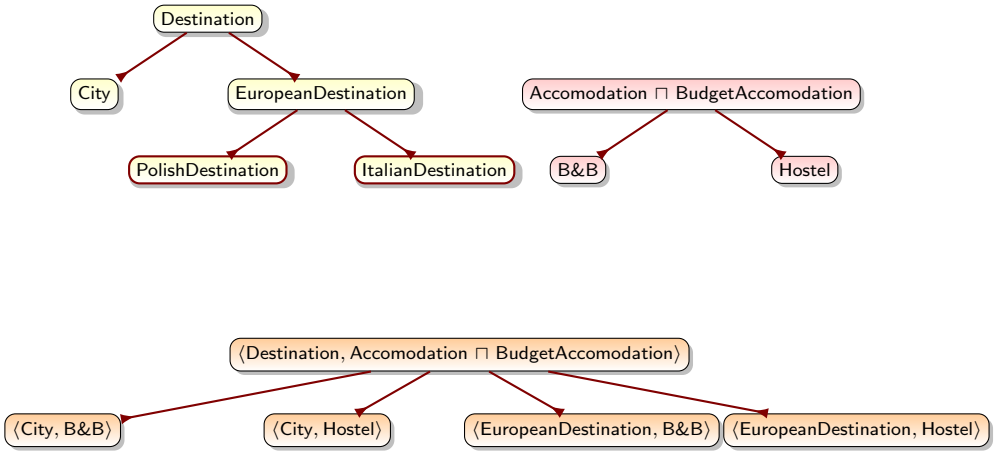


Fig. 3: Tree product illustration for query $Q_{2\_categorize\_by}$ from Example 4

leaves the membership check of instances to concepts in the tree nodes may be performed in order to find the most specific node under which to place the result tuple. Such incremental technique will allow to obtain more populated hierarchy in the longer time frame. This is in line with the recent trends in the Semantic Web research to propose the solutions whose completeness/quality increases over time. A sketch of the algorithm is shown in Figure 4. The time complexity of the

```
POPULATE(T, KB, H)
input:
T = (a_ij)_{i=1,r}^{j=1,n}: query answer table
KB: knowledge base
H: semantic category hierarchy
Q: query
output:
H: populated semantic category hierarchy

begin
    i ← 0
    while !RESOURCES_UP() and i < r
        a_i ← T.GETANSWER(I);
        ASSIGNTOMOSTSPECIFICSEMANTICCATEGORY(a_i, H, H.ROOT(), KB, Q);
return H
end

ASSIGNTOMOSTSPECIFICSEMANTICCATEGORY(a_i, H, H.NODE(), KB, Q)
begin
    ⟨(C_0), ..., (C_n)⟩ ← H.NODE().GETSEMANTICCATEGORY();
    assigned ← false;
    followsUnderNode ← true;
    j ← 0;
    while followsUnderNode and j < n
        a_ij ← a_i.GETBINDING(j);
        if ¬(KB ⊨ C(a_ij)) then
            followsUnderNode ← false;
        j ← j + 1;
    if followsUnderNode then
        for each H.NODE().CHILD()
            if ASSIGNTOMOSTSPECIFICSEMANTICCATEGORY(a_i, H, H.NODE().CHILD(), KB, Q)then
                assigned ← true;
        if assigned=false then
            H.NODE().ASSIGN(a_i);
            assigned ← true;
return assigned;
end
```

Fig. 4: Population of a semantic category hierarchy.

proposed population algorithm depends on a complexity of instance membership checks. The population algorithm itself has the complexity $O(nkb^m)$, where n represents the number of answers, k the number of grouping variables, and where b is the branching factor and m is the maximum depth of the tree. Note that besides this simple algorithm, also other solutions could be adopted, e.g. such in which a non-standard inference of computing *most specific concept* [11, 12] would be employed. It should be noted, hovewer, that for some description logics the most specific concept may not exist.

Table 2: Characteristics of test datasets.

dataset	DL	#concepts	#obj. roles	#individuals
NTN	$\mathcal{SHIF}$	49	29	728
FINANCIAL	$\mathcal{ALCIF}$	60	16	17941
VICODI	$\mathcal{ALHI}$	194	10	16942
LUBM	$\mathcal{SHI}(\mathbf{D})$	43	25	17174

Table 3: Tested queries.

dataset	query
NTN	$Q_1(x, y) = categorize\_by(x, y) : \mathsf{Agent}(x) \wedge \mathsf{relativeOf}(x, y)$
NTN	$Q_2(x, y, z) = categorize\_by(x, y, z) : \mathsf{CognitiveAgent}(x) \wedge \mathsf{collaboratesWith}(x, y) \wedge \mathsf{knows}(x, z)$
FINANCIAL	$Q_3(x, y) = categorize\_by(x, y) : \mathsf{Client}(x) \wedge \mathsf{hasCreditCard}(x, y) \wedge \mathsf{CreditCard}(y)$
FINANCIAL	$Q_4(x, y, z) = categorize\_by(x, y, z) : \mathsf{Client}(x) \wedge \mathsf{livesIn}(x, y) \wedge \mathsf{hasAgeValue}(x, z)$
VICODI	$Q_5(x, y) = categorize\_by(x, y) : \mathsf{Scientist}(x) \wedge \mathsf{hasRole}(y, x)$
VICODI	$Q_6(x, y, z) = categorize\_by(x, y, z) : \mathsf{Leader}(x) \wedge \mathsf{hasRole}(y, x) \wedge \mathsf{related}(x, z) \wedge \mathsf{Individual}(y) \wedge \mathsf{Location}(z)$
LUBM	$Q_7(x, y) = categorize\_by(x, y) : \mathsf{Faculty}(x) \wedge \mathsf{teacherOf}(x, y)$
LUBM	$Q_8(x, y, z) = categorize\_by(x, y, z) : \mathsf{Person}(x) \wedge \mathsf{memberOf}(x, y) \wedge \mathsf{subOrganizationOf}(x, z)$

5 Experimental evaluation

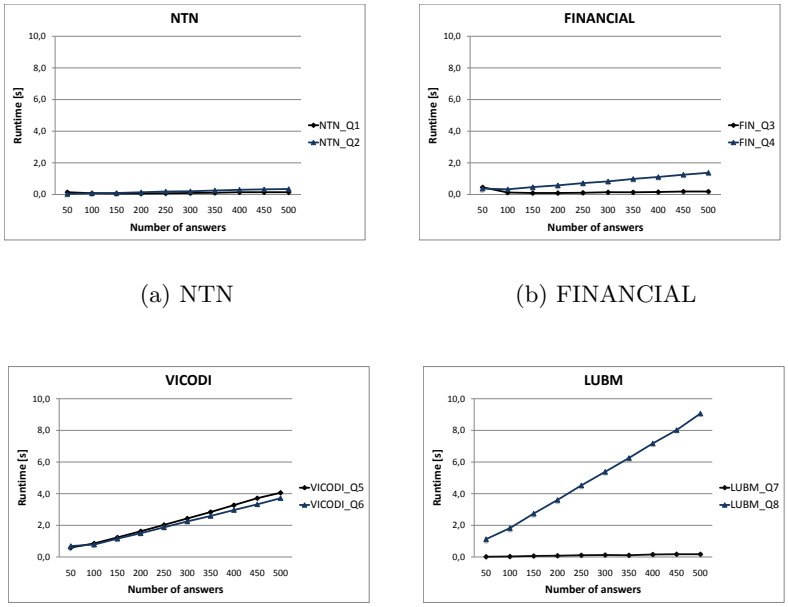
We have developed a proof-of-concept implementation of our approach in Java, on top of Pellet reasoner[5]. Please note, that state-of-the-art reasoners (including Pellet) do not support conjunctive queries with truly undistinguished variables. Therefore our prototype implementation supports only *DL-safe queries* [13], i.e. queries, where all variables are assumend to be bound to named individuals, and hence are treated as distinguished.

We have empirically tested our proposed approach using four benchmark datasets: NEW TESTAMENT-NAMES (NTN)[6], FINANCIAL[7], VICODI[8], and LUBM [9] (see: Table 2). The experiments were conducted on a laptop computer with a 1.67 GHz Intel processor, 1014 MB of RAM, running Windows Vista. For each of the datasets we tested 2 queries (see: Table 3). For each of the queries we tested how our method scales with the growing number of results for each query. The experimental results are shown in Figure 5. For each of the query we present the total time of computing, and populating a semantic category hierarchy.

Note that the Web users usually browse not more than a few dozens of the retrieved results. For this number of the results, the basic implementation of our proposed approach performes well, in real-time, on the tested cases, and scales linearly with the number of the processed results.

Note further, that not all the nodes of the generated semantic category hierarchy would become ultimately populated. This observation may lead to an optimization of the proposed algorithm for the cases where some precomputation would be possible to identify concepts without an extension. These concepts could be then stored on an index, and the nodes containing them removed from

[5] http://clarkparsia.com/pellet/
[6] NTN, http://protegewiki.stanford.edu/index.php/Protege_Ontology Library
[7] FINANCIAL, http://www.cs.put.poznan.pl/alawrynowicz/financial.owl
[8] VICODI, http://kaon2.semanticweb.org/download/test_ontologies.zip
[9] LUBM, http://swat.cse.lehigh.edu/projects/lubm/



(a) NTN (b) FINANCIAL

(c) VICODI (d) LUBM

Fig. 5: Results of the experiments.

the semantic category hierarchy in a preprocessing step, before population of the hierarchy, thus reducing the further complexity of the hierarchy population. We tested this technique experimentaly reaching a significant decrease in execution time (for the hardest query, Q_8, in case of processing 500 results, the execution time was shorter more than 5 times than this reported in the figure for the basic approach).

6 Related work

To the best of our knowledge, ours is the first proposal for grouping conjunctive query results on the ground of semantics of the underlying knowledge base.

While aggregate queries were extensively studied for relational databases, there are very few results for aggregate queries over ontologies, especially those that target to meet the pecularities of the KR formalisms of the Semantic Web. In [14] the syntax and semantics for epistemic aggregate queries over ontologies were proposed, and query answering for typical aggregate functions studied for an ontology language *DL-Lite*$_{\mathcal{A}}$. The work proposed there was motivated by the non-adequacy of certain answer semantics for conditional aggregate queries over

ontologies due to open world assumption. In [15] the meaning and implementation of grouping and aggregate queries over RDF graphs were studied, motivated by the drawback of the previous works where the grouping and aggregate operations had not take the graph structure of the base data into account.

However, none of the above approaches exploited the peculiar feature of the Semantic Web datasets, that is possible availability of the background ontologies expressing semantics of the data.

Some other works may also be considered as relevant to ours. In [16] an approach to automatically categorize the results of SQL queries has been proposed that consisted on a dynamic generation of a labeled, hierarchical category structure. Since this was proposed for relational database model, the constructed categories could only be built based on the values in the retrieved tuples, and not on any semantic information linked to them, due to the lack of background, domain knowledge. As relevant to this work the proposal for clustering the Semantic Web query results may be also considered, as proposed in [17, 18]. However, there is an essential difference between clustering query results, and our proposed semantic grouping, such that the former one involves *induction*, while the latter one is based simply on *deductive* reasoning.

7 Conclusions and Future Work

The paper proposes a method for the aggregation of concjuntive query results on the ground of an ontology of reference. We have defined a new type of *categorize by* queries for this purpose. To enable such queries, we have designed a method where in a deductive modality, answers are grouped taking into account dynamically generated subsumption hierarchy induced by the underlying knowledge base. This subsumption hierarchy has the form of a hierarchical, multi-valued classification that may support a faceted search of the results. The research presented in the paper may constitute a proposal for the next generation query languages oriented to knowledge bases annotated with ontology languages.

To the best of our knowledge, this work is the first to propose grouping of conjunctive query results on the ground of semantics of the underlying knowledge base. As such it may be an important first step towards the proposed direction of semantic grouping, and inspiration to conduct future work on the proposed idea. In particular, in the future, alternative versions of the components of the proposed method may be investigated such as for example different tree/graph product operators, or different techniques of assigning answers to semantic groups. The interesting future research may also concern an investigation on a possible integration of semantic grouping into query processing, and query execution plans.

References

1. Calvanese, D., De Giacomo, G., Lenzerini, M.: On the decidability of query containment under constraints. In: Proc. of the 17th ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'98). (1998) 149–158

2. Horrocks, I., Tessaris, S.: Querying the semantic web: a formal approach. In: Proc. of the 13th Int. Semantic Web Conf. (ISWC 2002), number 2342 in Lecture Notes in Computer Science, Springer-Verlag (2002) 177–191
3. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In Doherty, P., Mylopoulos, J., Welty, C., eds.: Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06)., AAAI Press (2006)
4. Ortiz, M., Calvanese, D., Eiter, T.: Data complexity of answering unions of conjunctive queries in shiq. In Parsia, B., Sattler, U., Toman, D., eds.: Proceedings of the International Workshop on Description Logics (DL'06). Volume 189 of CEUR-WS., CEUR (2006)
5. Glimm, B., Horrocks, I., Lutz, C., Sattler, U.: Conjunctive query answering for the description logic shiq. J. Artif. Int. Res. **31**(1) (2008) 157–204
6. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook. Cambridge University Press (2003)
7. OWL: Web Ontology Language Reference Version 1.0 (2003) http://www.w3.org/TR/owl-ref.
8. Badea, L., Nienhuys-Cheng, S.H.: A refinement operator for description logics. In Cussens, J., Frisch, A.M., eds.: ILP. Volume 1866 of Lecture Notes in Computer Science., Springer (2000) 40–59
9. Iannone, L., Palmisano, I., Fanizzi, N.: An algorithm based on counterfactuals for concept learning in the semantic web. Appl. Intell. **26**(2) (2007) 139–159
10. Lehmann, J., Hitzler, P.: A refinement operator based learning algorithm for the $\mathcal{ALC}$ description logic. In Blockeel, H., et al., eds.: Proceedings of the 17th International Conference on Inductive Logic Programming, ILP2007. Volume 4894 of LNCS., Springer (2008) 147–160
11. Baader, F., Küsters, R.: Computing the least common subsumer and the most specific concept in the presence of cyclic aln-concept descriptions. In Herzog, O., Günter, A., eds.: KI. Volume 1504 of Lecture Notes in Computer Science., Springer (1998) 129–140
12. Küsters, R., Molitor, R.: Approximating most specific concepts in description logics with existential restrictions. In Baader, F., Brewka, G., Eiter, T., eds.: KI/ÖGAI. Volume 2174 of Lecture Notes in Computer Science., Springer (2001) 33–47
13. Motik, B., Sattler, U., Studer, R.: Query answering for OWL-DL with rules. Journal of Web Semantics: Science, Services and Agents on the World Wide Web **3**(1) (2005) 41–60
14. Calvanese, D., Kharlamov, E., Nutt, W., Thorne, C.: Aggregate queries over ontologies. In: ONISW '08: Proc. of the 2nd international workshop on Ontologies and Information systems for the Semantic Web, New York, NY, USA, ACM (2008) 97–104
15. Seid, D., Mehrotra, S.: Grouping and aggregate queries over semantic web databases. In: International Conference on Semantic Computing, Los Alamitos, CA, USA, IEEE Computer Society (2007) 775–782
16. Chakrabarti, K., Chaudhuri, S., Hwang, S.w.: Automatic categorization of query results. In: SIGMOD '04: Proc. of the 2004 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM (2004) 755–766
17. Ławrynowicz, A.: Grouping results of queries to ontological knowledge bases by conceptual clustering. In Nguyen, N.T., Kowalczyk, R., Chen, S.M., eds.: ICCCI. Volume 5796 of Lecture Notes in Computer Science., Springer (2009) 504–515

18. Ławrynowicz, A.: Query results clustering by extending sparql with cluster by. In Meersman, R., Herrero, P., Dillon, T.S., eds.: OTM Workshops. Volume 5872 of Lecture Notes in Computer Science., Springer (2009) 826–835