Towards a Cooperative Query Language for Semantic Web Database Queries

Stéphane Jean, Allel Hadjali, and Ammar Mars

LIAS/ISAE-ENSMA - University of Poitiers
1, Avenue Clement Ader, 86960 Futuroscope Cedex, France
{stephane.jean,allel.hadjali,ammar.mars}@ensma.fr

Abstract. In this paper, we address the problem of cooperative query answering on Semantic Web Databases ($\mathcal{SWDB}s$) in order to overcome the problem of empty or unsatisfactory answers. While most existing approaches propose an automatic relaxation process based on RDF-S entailment, our proposition consists in endowing $\mathcal{SWDB}$ query languages with relaxation and approximation operators. The main interest of this approach is that the relaxation process can be finely controlled by the user or implicitly by the system. Experiments on real data are conducted to evaluate (1) the scalability of the proposed operators and (2) the interest of the similarity measure defined to rank the alternative answers.

1 Introduction

Semantic Web Databases ($\mathcal{SWDB}s$) have been designed to provide an efficient management of semantic data. As users are often not aware of the contents of the queried $\mathcal{SWDB}s$, their queries may return an empty set of answers. Query relaxation is one cooperative technique proposed to solve this problem. It aims at expanding the scope of a query so that more information can be gathered in the answers. As the main novelty in this context is the fact that $\mathcal{SWDB}s$ store ontologies to explicit the semantics of data, one can leverage such ontologies for query relaxation. As illustration, let us consider a database of hotels (*Hotels-base*) borrowed from the site `http://www.hotelsbase.org/`. These hotels are organized in a hierarchy of different types (see Figure 1) and described by a set of properties such as *price*, *stars* or *rating*. If a user wants to find a *Bed & Breakfast* hotel in Paris under 90 euros, (s)he will get an empty answer. This query can be relaxed either by weakening the predicate *price* or by changing the type of hotel desired by leveraging the subsumption relationships defined in the ontology. To choose the most relevant relaxation techniques that should be applied, most approaches use a function for ranking the possible relaxed queries. If this solution presents the advantage of being completely automatic, it relies heavily on the ranking function used. Moreover, this function can be inappropriate for the targeted domain. Another drawback of the above approaches is that the relaxation process can not be controlled. If we come back to our query example, a user may want to relax his/her price condition up to 100 euros and, if there is still no answer, change the type of hotels desired. In order to not

R. Meersman et al. (Eds.): OTM 2013, LNCS 8185, pp. 519–526, 2013.

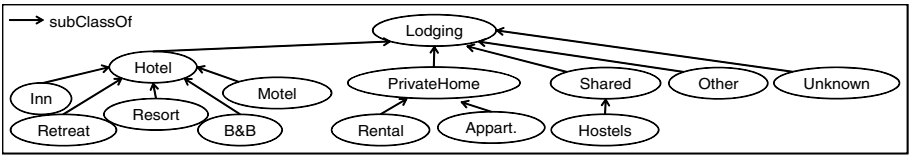

Fig. 1. Ontology example based on Hotelsbase

violate this intention, the system designer needs a tool to guide and control the relaxation process.

In this paper, we propose to extend the query language of $\mathcal{SWDB}s$. This extension consists of three operators that can be combined to finely relax an $\mathcal{SWDB}$ query and control the relaxation process according to the user preferences and/or the targeted domain [1]. These operators belong to two families: predicate and conceptual relaxations. In the former, we operate only on the (fuzzy/crisp) predicates involved in the query at hand by applying some tolerance borrowed from fuzzy set theory [2]. As for the latter, we leverage a domain ontology to substitute a concept present in the failing query by another concept by using hierarchical relations. A similarity measure is also defined to rank-order the answers of a relaxed or approximate query and return the top-k answers. The feasibility of our proposition is shown by describing its implementation on the OntoDB $\mathcal{SWDB}$.

This paper is organized as follows. Section 2 presents the three proposed operators while Section 3 shows how they can be used in an $\mathcal{SWDB}$ query language. Section 4 provides some experiments conducted to show the feasibility of our proposal and evaluate the scalability of our operators on real data. Section 5 discusses related works and Section 6 concludes the paper.

2 $\mathcal{SWDB}$ Query Approximation Operators

We consider ontologies defined as a set of classes C and properties P expressed in an ontology formalism such as RDF-S. $\mathcal{SWDB}$ queries are expressed as a graph pattern with $FILTER$ conditions composed of crisp or fuzzy predicates (fuzzy predicates are modeled thanks to trapezoidal membership functions expressed by the quadruplet (a, b, c, d) where $[b, c]$ (resp. $]a, d[$) stands for the core (resp. support) of the predicates).

2.1 The *PRED* Operator

$Signature : PRED : Q \times Pred \times Integer \times Interval \rightarrow Q$.
$PRED(q, p, \epsilon, i)$ relaxes the predicate p in the query q using a tolerance value of ϵ until the answer of the revised query is not empty or when the support is not included in the interval of validity i (see [3] for more details). The last two parameters are optional. If they are not specified, the tolerance value is set to

$\epsilon = 0.1$ and the generic interval of validity $V = [(\sqrt{5}-1)/2, (\sqrt{5}+1)/2]$ that constraints the tolerance relation of interest (expressed thanks to a particular closeness relation), is used [4].

Example. Let us consider the following query:
(?id, type, B&B) (?id, stars, 5)(?id, cityname, Belgrade) (?id, price, ?price)
FILTER **(?price, moderate)**APPROX PRED(?price)
where the semantics of the fuzzy predicate *moderate* is given by $(45, 50, 60, 65)$. As this query fails, the predicate *moderate* is relaxed by $T^{\uparrow}(moderate) = (40, 50, 60, 71.33)$ for a tolerance value $\epsilon = 0.1$. This process can be repeated n times until enough answers are found or when the interval of validity V is overstepped.

2.2 The *GEN* Operator

Signature : $GEN : Q \times C \cup P \times C \cup P \times Integer \rightarrow Q$
$GEN(q, c, c', i)$ relaxes the query q by replacing the class c by one of its superclasses c' and dropping the triples corresponding to properties not defined on c'. The parameter i specifies the maximum number of levels in the hierarchy that must be considered. The two last parameters are optional. If they are not defined, the superclass is determined using a similarity function and the process is repeated until the root class is reached (or when the result is satisfying). Considering the similarity function, both distance-based measure and information content measure have been used in previous work [5,6]. We have chosen the information content measure defined in [7] as it was more relevant to our data:

$$sim(c, c') = \frac{IC(msca(c,c'))}{IC(c)+IC(c')-IC(msca(c,c'))}$$

where $msca(c, c')$ is the concept that subsumes the two concepts being compared and $IC(c) = -logPr(c)$ corresponds to the information content of the class c which is defined according to the probability $Pr(c)$ of getting an instance of the class c in the $\mathcal{SWDB}$.

Example. Let us consider the following query: "retrieve five star Bed&Breakfast Hotels located in the city of Belgrade with a price less than 29":
(?id, type, B&B) (?id, stars, 5) (?id, cityname, Belgrade)
(?id, price, ?price) FILTER (?price <= 29) APPROX GEN(B&B)
This failing query is relaxed using the unique superclass of $B\&B$ i.e., $Hotel$. This relaxed query returns 1 result and thus the relaxation process stops.

2.3 The *SIB* Operator

Signature : $SIB : Q \times C \times 2^C \rightarrow Q$.
$SIB(q, c, [c_1, \cdots, c_n])$ modifies the query q by replacing the class c by its sibling class c_1. If the query does not return the desired answer, the siblings classes $c_2 \cdots c_n$ are successively used. The list of sibling classes is an optional parameter. If it is not defined, all the siblings classes are used sorted by similarities with c.

Example. Let us consider the following query: "retrieve Resort Hotels located in the city of Istanbul whose rating is at least 8":
(?id,type,Resort) (?id, cityname, Istanbul) (?id, rating, ?rating)
FILTER (?rating >= 8) APPROX SIB(Resort, [B&B, Inn])
This failing query is relaxed with the sibling class $B\&B$. This relaxed query returns 2 results and the relaxation process stops.

3 $\mathcal{SWDB}$ Query Language Extension

The proposed extension of $\mathcal{SWDB}$ query languages has been illustrated by several examples in the previous section. We define here precisely this extension and introduce our ranking function of approximate answers.

3.1 Syntax of the Approx Clause

The proposed extension of $\mathcal{SWDB}$ query languages consists of a new clause $APPROX$. The syntax of this clause is as follows.

⟨*approx clause*⟩ ::= APPROX ⟨*approx operator expr*⟩ [TOP ⟨*integer*⟩]
⟨*approx operator*⟩ ::= ⟨*pred operator*⟩ | ⟨*gen operator*⟩ | ⟨*sib operator*⟩
⟨*pred operator*⟩ ::= PRED (⟨*var*⟩[,real][,⟨*interval*⟩])
⟨*gen operator*⟩ ::= GEN (⟨*literal*⟩[,⟨*literal*⟩][,⟨*integer*⟩])
⟨*sib operator*⟩ ::= SIB (⟨*literal*⟩ [,[⟨*literal*⟩(,⟨*literal*⟩)*]])

This clause is used to specify the number of results expected by the user (the default value is 1) and the operators that must be applied to obtain these answers if the result of the initial query is empty. The previously presented operators can be combined (⟨*approx operator expr*⟩) by the AND and OR constructs whose semantics is defined in the next section.

3.2 Semantics of the *Approx* Clause

The $PRED$, GEN and SIB operators previously defined can be combined with the two following constructs. (1) AND: the two approximations expressed by the two operands are done simultaneously. (2) OR: the approximation expressed by the first operand is done and if a satisfying answer is still not found, the query is relaxed using the second operand. Formally, if Op_1 and Op_2 are two operators of relaxation or approximation that can be repeated m and n times respectively:

$$Op_1(q)\ AND\ Op_2(q) \equiv Op_1(Op_2(q))$$
$$Op_1(q)\ OR\ Op_2(q) \equiv Op_1(q)\ if\ |Answers(Op_1(q))| >= k\ else\ Op_2(q)$$

As individual operator, the approximation process resulting of a combination of operators can be repeated several times:

$$(Op_1(q)\ AND\ Op_2(q))^{(i)} \equiv Op_1^{(i)}(Op_2^{(i)}(q))\ where\ i < min(m,n)$$
$$(Op_1(q)\ OR\ Op_2(q))^{(i)} \equiv Op_1^{(i)}(q)\ if\ i < m\ else\ Op_2^{(i-m)}(q)$$

3.3 Distance of the Approximate Answers with the Original Query

Let us consider the following $\mathcal{SWDB}$ query q:
(?h, type, B&B) (?h, price, ?price) FILTER(?price is moderate)
APPROX GEN(B&B, Hotel) AND PRED(?price, 0.05)
where *moderate* is defined by the quadruplet (48, 50, 55, 57). If this query fails, the relaxed query will use a variant of *moderate* given by $p = (45.5, 50, 55, 59.8)$. We need to assess the extent to which each answer of q', denoted h_i, satisfies the original query q. In previous works [5,6], the proposed measures of satisfaction take only into account the similarity between q and q': the satisfaction degree of an answer is the maximum score among all of the relaxed queries it matches. Thus, direct instances of the class $Hotel$ and direct instances of a subclass of $Hotel$ (e.g., $Motel$), that are results of the query q', will have the same degree of satisfaction. The same score will also be given to other hotels even if their prices are quite different. To solve this problem and return the top-k results more accurately, our proposed measure of satisfaction takes into account both the similarity between q and q' and the satisfaction of h_i w.r.t q':

$$SatQ(h_i) = min(Sim(q', q), SatQ'(h_i)) \quad (1)$$

Where $Sim(q', q)$ is the similarity measure between q' and q, and $SatQ(h_i)$ stands for the satisfaction degree of h_i of the query q. Formula (1) means that the satisfaction $SatQ(h_i)$ is the least degree between $Sim(q', q)$ and $SatQ'(h_i)$. This pessimistic attitude is expressed by the min operator.

Similarity Measure Computing. $Sim(q', q)$ is the aggregation of the similarity of each triple of q' with its corresponding triple in q. Thus, we have:
$Sim(q', q) = min(Sim(Hotel, B\&B), Sim(p, moderate))$
We have seen in Section 2.2 how to compute $Sim(Hotel, B\&B)$. For $Sim(p, moderate)$, the distance of Hausdorff [8] can be used:
$Sim(p, moderate) = 1/(1 + Dist(p, moderate))$
See [9] for more details on the application of the distance of Hausdorff to fuzzy sets. In our example, we obtain $Dist(p, moderate) = 0.8$.

Computation of the Satisfaction Degree. The calculus of $SatQ'(h_i)$ boils down to compute the satisfaction degree of h_i w.r.t to the gradual predicate p and its type. Arguably the degree of satisfaction of an instance of a subclass of $Hotel$ (e.g., Inn) w.r.t q' should not be the same than a direct instance of $Hotel$. As a consequence, we compute $SatQ'(h_i)$ as follows:

$$SatQ'(h_i) = min(\max_{t \in directType(h_i)} Sim(t, Hotel), \mu_p(h_i.price))$$

where $directType$ is the set of the most specific classes an instance belongs to.

To show the interest of our proposed measure, table 1 presents the satisfaction degree of several approximative answers of our illustrative query q. This example shows that the ranking of our measure ($SatQ(h_i)$) is more precise than a ranking based only on $Sim(Q, Q')$. As $Sim(B\&B, Hotel) > Sim(B\&B, C_{sub})$ $\forall C_{sub} \in subClassOf(Hotel)$, the priority is given to the direct instances of $Hotel$ and

then, to instances of *Resort* (the closest to *Hotel*), and so on. As most hotels are classified in the generic class *Hotel* (see section 4), the similarities between the different classes are quite low. And, since we use a pessimistic approach (expressed by the *min* operator), the ranking of the answers is dominated by the type rather than the price. This fact is illustrated by the answers h_1 and h_2 which have the same ranking even if they have different prices. This problem could be solved by using another operator of aggregation such as the product.

Table 1. Satisfaction degree of approximate answers (dt = DirectType)

h_i	$Sim(Q, Q')$	$SatQ'(h_i)$	$SatQ(h_i)$
$h_1, dt = Hotel, price = 58$	0.09	0.37	0.09
$h_2, dt = Hotel, price = 57$	0.09	0.58	0.09
$h_3, dt = Resort, price = 59$	0.09	0.08	0.08
$h_4, dt = Retreat, price = 58$	0.09	0.04	0.04
$h_5, dt = Inn, price = 59$	0.09	0.07	0.07

4 Preliminary Experimentation

We have implemented our relaxation techniques on the OntoDB Semantic Database [10] and evaluated the performance of each proposed operator on the dataset provided by Hotelsbase. This dataset is composed of 500K hotels distributed as follow: *Hotel* (370361), *B&B* (18452), *Inn* (6560), *Retreat* (539), *Resort* (14410) and *Motel* (8853). For each conceptual relaxation technique, we have done two implementations: (1) a straight implementation (2) an optimized version where the ontology is put in main memory and thus do not need to be read in the database during the relaxation algorithm. We have considered a set of queries that requires only one relaxation step and we have compared the cost of the initial query with the one of the relaxed query. Due to space limitation, we only provide results about vertical relaxation.

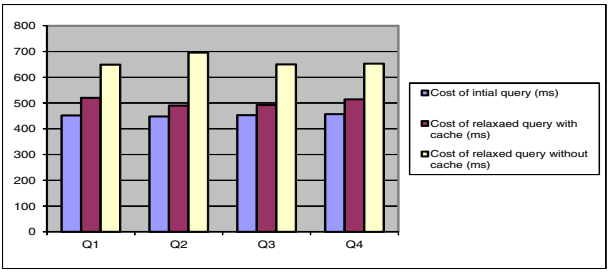

Fig. 2. Results of the experiments on vertical relaxation

Vertical Relaxation (VR). Four failing queries are considered (see below). In the right part of Figure 2, we provide the execution time of each query and its relaxed variant according to the implementation with cache (IWC) and the implementation without cache (IWOC).

Q_1 : (?id, type, B&B) (?id, stars, 5) (?id, cityname, Belgrade)
(?id, price, ?price) FILTER (?price <= 29) APPROX GEN (B&B)
Q_2 : (?id, type, INN) (?id, stars, 5) (?id, cityname, Shanghai)
(?id, price, ?price) FILTER (?price <= 15) APPROX GEN (INN)
Q_3 : (?id, type, Motel) (?id, stars, 5) (?id, cityname, Istanbul)
(?id, price, ?price) FILTER (?price <= 80) APPROX GEN (Motel)
Q_4 : (?id, type, Resort)(?id, rating, 3)(?id, stars, 3)(?id, state, California)
(?id, price, ?price) FILTER (?price < 85) APPROX GEN (Resort)

The results reported in Figure 2 illustrate the feasibility of the conceptual relaxation approach proposed, in the one hand, and confirm the claim that IWC is better than IWOC, on the other hand. One can also observe that the extra cost resulting from the execution of relaxed queries in IWC is not very important (the average of this cost amounts to *56 ms* for one relaxation step). This means that if the ontology hierarchy contains a maximum of 10 levels, the relaxation process does not exceed 5s (500ms x 10 where the average execution time of a query is 450 ms).

5 Related Work

Hurtado *et al.* [11] have introduced the relaxation of RDF queries through RDF-S entailment. This work has been extended in [12] to combine query approximation and relaxation and to consider conjunctive regular path queries. We note that the types of relaxation encompassed by these techniques do not include a fine-grained relaxation of filters (a value in a triple can only be replaced by a variable and not by an approximate value). Moreover few parameters are available in the proposed operators to control precisely the relaxation or approximation process. Dolog *et al.* [13] proposed a method for automatically relaxing over-constrained RDF queries based on background knowledge about the domain model and user preferences. The type of relaxation proposed consists in rewriting the original query by replacing some of its parts by applying specific rules. For instance, replacing a highly preferred value by a less preferred one. Compared to our approach, this approach requires to define domain and user preferences and do not use similarity relations between concepts of the ontology. Huang *et al.* have addressed the problem of relaxing RDF queries in [6] and [5]. These approaches rely on the relaxation model of [11]. (with the same limitation to relax filters) Two particular contributions are made in these papers: (i) ranking the relaxed queries using a distance-based method [6] or a measure based on information content [5] and (ii) optimizing the relaxation process to retrieve the top-k answers. We have shown in Section 3.3 that our proposed measure refines the one proposed in these papers.

6 Concluding Remarks

This paper addressed the need to control and guide the relaxation process when a $\mathcal{SWDB}$ query fails. Our proposition consists of three cooperative operators that can be called and combined in an $\mathcal{SWDB}$ query language such as SPARQL. In addition to these operators, we have defined a similarity measure to compute the distance between an approximate answer and the original query. The originality of this measure is to take into account not only the similarity between the initial and relaxed queries but also the degree of satisfaction of an answer to the relaxed query. By this way, one can distinguish between the answers of a relaxed query and thus obtain for instance the top-k approximate answers. To show the feasibility of our approach, we have implemented it on the OntoDB $\mathcal{SWDB}$ and we have made some preliminaries experiments to test the scalability of the three proposed operators. As for future work, we plan to achieve more extensive experimentations on the efficiency and effectiveness of our operators.

References

1. Tapucu, D., Jean, S., Ait-Ameur, Y., Unalir, M.O.: An extention of ontology based databases to handle preferences. In: Proc. ICEIS 2009, pp. 208–214 (2009)
2. Zadeh, L.A.: Fuzzy sets. Information and Control 8(3), 338–353 (1965)
3. Bosc, P., Hadjali, A., Pivert, O.: Incremental controlled relaxation of failing flexible queries. Journal of Intelligent Information Systems 3(3), 261–283 (2009)
4. Hadjali, A., Dubois, D., Prade, H.: Qualitative reasoning based on fuzzy relative orders of magnitude. IEEE Transactions on Fuzzy Systems 1(1), 9–23 (2003)
5. Huang, H., Liu, C., Zhou, X.: Approximating query answering on RDF databases. Journal of World Wide Web 15(1), 89–114 (2012)
6. Huang, H., Liu, C., Zhou, X.: Computing Relaxed Answers on RDF Databases. In: Bailey, J., Maier, D., Schewe, K.-D., Thalheim, B., Wang, X.S. (eds.) WISE 2008. LNCS, vol. 5175, pp. 163–175. Springer, Heidelberg (2008)
7. Pirró, G., Euzenat, J.: A feature and information theoretic framework for semantic similarity and relatedness. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 615–630. Springer, Heidelberg (2010)
8. Nutanong, S., Jacox, E.H., Samet, H.: An incremental hausdorff distance calculation algorithm. Proceedings of VLDB (PVLDB 2011) 4(8), 506–517 (2011)
9. Chaudhuri, B.B., Rosenfeld, A.: A modified hausdorff distance between fuzzy sets. Journal of Information Sciences 118(1-4), 159–171 (1999)
10. Jean, S., Dehainsala, H., Xuan, D.N., Pierra, G., Bellatreche, L., Aït-ameur, Y.: OntoDB: It is Time to Embed your Domain Ontology in your Database. In: Kotagiri, R., Radha Krishna, P., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 1119–1122. Springer, Heidelberg (2007)
11. Hurtado, C.A., Poulovassilis, A., Wood, P.T.: Query Relaxation in RDF. In: Spaccapietra, S. (ed.) Journal on Data Semantics X. LNCS, vol. 4900, pp. 31–61. Springer, Heidelberg (2008)
12. Poulovassilis, A., Wood, P.T.: Combining Approximation and Relaxation in Semantic Web Path Queries. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 631–646. Springer, Heidelberg (2010)
13. Dolog, P., Stuckenschmidt, H., Wache, H., Diederich, J.: Relaxing RDF queries based on user and domain preferences. Journal of Intelligent Information System 33(3), 239–260 (2009)