Top-k Retrieval in Description Logic Programs Under Vagueness for the Semantic Web

Thomas Lukasiewicz[1,2] and Umberto Straccia[3]

[1] DIS, Sapienza Università di Roma, Via Ariosto 25, I-00185 Roma, Italy
lukasiewicz@dis.uniroma1.it
[2] Institut für Informationssysteme, Technische Universität Wien,
Favoritenstraße 9-11, A-1040 Wien, Austria
lukasiewicz@kr.tuwien.ac.at
[3] ISTI-CNR, Via G. Moruzzi 1, I-56124 Pisa, Italy
straccia@isti.cnr.it

Abstract. Description logics (DLs) and logic programs (LPs) are important representation languages for the Semantic Web. In this paper, we address an emerging problem in such languages, namely, the problem of evaluating ranked top-k queries. Specifically, we show how to compute the top-k answers in a data-complexity tractable combination of DLs and LPs under vagueness.

1 Introduction

Description logics (DLs) and logic programs (LPs) are important representation languages for the Semantic Web. In this paper, we address an emerging issue, namely, the problem of evaluating ranked top-k queries in a combination of such languages under vagueness. Under the classical semantics, an answer to a query is a set of tuples that satisfy a query. The information need of a user, however, very often involves so-called *vague predicates*. For instance, in a logic-based e-commerce process, we may ask "find a car costing *around* \$15000" (see [12]); or in ontology-mediated access to multimedia information, we may ask "find images *about* cars, which are *similar* to a given one" (see, e.g., [11,19]). Unlike the classical case, tuples now satisfy these queries to a degree (usually in $[0, 1]$). Therefore, a major problem is that now an answer is a set of tuples *ranked* according to their degree. This poses a new challenge when we have to deal with a huge amount of facts. Indeed, virtually every tuple may satisfy a query with a non-zero degree, and thus has to be ranked. Of course, computing all these degrees, ranking them, and then selecting the top-k ones is likely not feasible in practice.

In this work, we address the top-k retrieval problem for a data complexity tractable combination of DLs and LPs under a many-valued semantics. In our language, at the extensional level, each fact may have a truth value, while at the intensional level many-valued DL axioms and LP rules describe the application domain.

2 Preliminaries

The truth space that we consider here is the finite set $[0, 1]_m = \{\frac{0}{m}, \frac{1}{m}, \ldots, \frac{m-1}{m}, \frac{m}{m}\}$ (for a natural number $m > 0$), which is pretty common in fuzzy logic. Throughout the

H. Prade and V.S. Subrahmanian (Eds.): SUM 2007, LNAI 4772, pp. 16–30, 2007.

ID	MODEL	TYPE	PRICE	KM	COLOR	AIRBAG	INTERIOR TYPE	AIR COND	ENGINE FUEL
455	MAZDA 3	Sedan	12500	10000	Red	0	VelvetSeats	1	Gasoline
34	ALFA 156	Sedan	12000	15000	Black	1	LeatherSeats	0	Diesel
1812	FORD FOCUS	StationVagon	11000	16000	Gray	1	LeatherSeats	1	Gasoline

Fig. 1. The car table

ID	HOTEL	PRICE Single	PRICE Double	DISTANCE	s
1	Verdi	100	120	5Min	0.75
2	Puccini	120	135	10Min	0.5
3	Rossini	80	90	15Min	0.25

Fig. 2. The hotel table

paper, we assume $m = 100$ in the examples with usual decimal rounding (e.g., 0.375 becomes 0.38, while 0.374 becomes 0.37).

A *knowledge base* $\mathcal{K}$ consists of a *facts component* $\mathcal{F}$, a *DL component* $\mathcal{O}$, and an *LP component* $\mathcal{P}$, which are all three defined below.

Facts Component. $\mathcal{F}$ is a finite set of expressions of the form

$$\langle R(c_1, \ldots, c_n), s\rangle \ ,$$

where R is an n-ary relation, every c_i is a constant, and s is a degree of truth (or simply *score*) in $[0,1]_m$. For each R, we represent the facts $\langle R(c_1, \ldots, c_n), s\rangle$ in $\mathcal{F}$ by means of a relational $n+1$-ary table T_R, containing the records $\langle c_1, \ldots, c_n, s\rangle$. We assume that there cannot be two records $\langle c_1, \ldots, c_n, s_1\rangle$ and $\langle c_1, \ldots, c_n, s_2\rangle$ in T_R with $s_1 \neq s_2$ (if there are, then we remove the one with the lower score). Each table is sorted in descending order with respect to the scores. For ease, we may omit the score component and in such cases the value 1 is assumed.

Example 1 ([12]). Suppose we have a car selling site, and we would like to buy a car. The cars belong to the relation $CarTable$ shown in Fig. 1. Here, the score is implicitly assumed to be 1 in each record. For instance, the first record corresponds to the fact

$$\langle CarTable(455, MAZDA3, Sedan, 12500, 10000, Red, 0, VelvetSeats, 1, Gasoline), 1\rangle \ .$$

Example 2 ([15,20]). Suppose we have information about hotels and their degree of closeness to the city center, computed from the walking distance according to some predefined function, and we would like to find a cheap hotel close to the city center. The hotels belong to the relation $CloseHotelTable$ shown in Fig. 2. The column s indicates the degree of closeness. For instance, the first record corresponds to the fact

$$\langle CloseHotelTable(1, Verdi, 100, 120, 5Min), 0.75\rangle \ .$$

Semantically, an *interpretation* $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}}\rangle$ consists of a *fixed infinite domain* Δ and an *interpretation function* $\cdot^{\mathcal{I}}$ that maps every n-ary relation R to a partial function $R^{\mathcal{I}}\colon \Delta^n \to [0,1]_m$ and every constant to an element of Δ such that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$ (unique name assumption). We assume to have one object for each constant, denoting exactly that object. In other words, we have standard names, and we do not distinguish

$Cars \sqsubseteq Vehicles$	$LeatherSeats \sqsubseteq Seats$
$Trucks \sqsubseteq Vehicles$	$VelvetSeats \sqsubseteq Seats$
$Vans \sqsubseteq Vehicles$	$MidSizeCars \sqsubseteq PassengerCars$
$LuxuryCars \sqsubseteq Cars$	$SportyCars \sqsubseteq PassengerCars$
$PassengerCars \sqsubseteq Cars$	$CompactCars \sqsubseteq PassengerCars$
$\exists 1{:}CarTable \sqsubseteq Cars$	$Vehicles \sqsubseteq \exists 1{:}hasMaker$
$Sedan \sqsubseteq Cars$	$Vehicles \sqsubseteq \exists 1{:}hasPrice$
$StationWagon \sqsubseteq Cars$	$\exists 1{:}hasPrice \sqsubseteq Vehicles$
$\exists 9{:}CarTable \sqsubseteq Seats$	$\exists 1{:}hasMaker \sqsubseteq Vehicles$
$Mazda \sqsubseteq CarMake$	$\exists 2{:}hasMaker \sqsubseteq CarMaker$
$AlfaRomeo \sqsubseteq CarMake$	$Cars \sqsubseteq \exists 1{:}hasKm$
$Ford \sqsubseteq CarMake$	$\exists 2{:}hasFuel \sqsubseteq FuelType$

Fig. 3. A car selling ontology

between the alphabets of constants and the objects in Δ. Note that, since $R^{\mathcal{I}}$ may be a partial function, some tuples may not have a score. Alternatively, we may assume $R^{\mathcal{I}}$ to be a total function. We use the former formulation to distinguish the case where a tuple $\boldsymbol{c}$ may be retrieved, even though the score is 0, from the case where a tuple is not retrieved, since it does not satisfy the query. In particular, if a tuple does not belong to an extensional relation, then its score is assumed to be undefined, while if $R^{\mathcal{I}}$ is total, then the score of this tuple would be 0.

An interpretation $\mathcal{I}$ is a *model* of (or *satisfies*) a fact $\langle R(c_1, \ldots, c_n), s\rangle$, denoted $\mathcal{I} \models \langle R(c_1, \ldots, c_n), s\rangle$, iff $R^{\mathcal{I}}(c_1, \ldots, c_n) \geqslant s$ whenever $R^{\mathcal{I}}(c_1, \ldots, c_n)$ is defined.

DL Component. $\mathcal{O}$ is a finite set of *axioms* having the form

$$C_1 \sqcap \ldots \sqcap C_l \sqsubseteq C$$

(called *concept inclusion*), where all C_i and C are concept expressions. Informally, $C_1 \sqcap \ldots \sqcap C_l \sqsubseteq C$ says that if c is an instance of C_i to degree s_i, then c is an instance of C to degree at least $\min(s_1, \ldots, s_l)$. A concept expression is either an atomic concept A or of the form $\exists i{:}R$, where R is an n-ary relation and $i \in \{1, \ldots, n\}$. Informally, $\exists i{:}R$ is the projection of R on the i-th column. These concepts are inspired by the description logic *DLR-Lite* [2], a LogSpace data complexity family of DL languages, but still with good representation capabilities.

We recall that despite the simplicity of its language, the DL component is able to capture the main notions (though not all, obviously) to represent structured knowledge. In particular, the axioms allow us to specify *subsumption*, concept A_1 is subsumed by concept A_2, using $A_1 \sqsubseteq A_2$; *typing*, using $\exists i{:}R \sqsubseteq A$ (the i-th column of R is of type A); and *participation constraints*, using $A \sqsubseteq \exists i{:}R$ (all instance of A occur in the projection of R on the i-th column).

Example 3. Consider again Example 1. An excerpt of the domain ontology is described in Fig. 3 and partially encodes the web directory behind the car selling site `www.autos.com`. For instance, the axiom

$$Vehicles \sqsubseteq \exists 1{:}hasPrice$$

dictates that each vehicle has a price.

Semantically, an interpretation $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$ maps every atom A to a partial function $A^{\mathcal{I}} \colon \Delta \to [0,1]_m$. In the following, $\boldsymbol{c}$ denotes an n-tuple of constants, and $\boldsymbol{c}[i]$ denotes the i-th component of $\boldsymbol{c}$. Then, $\cdot^{\mathcal{I}}$ has to satisfy, for all $c \in \Delta$:

$$(\exists i{:}R)^{\mathcal{I}}(c) = \sup_{\boldsymbol{c'} \in \Delta^n,\, \boldsymbol{c'}[i]=c,\ R^{\mathcal{I}}(\boldsymbol{c'}) \text{ is defined}} R^{\mathcal{I}}(\boldsymbol{c'}) \,.$$

Then, $\mathcal{I} \models C_1 \sqcap \ldots \sqcap C_l \sqsubseteq C$ iff, for all $c \in \Delta$, $\min({C_1}^{\mathcal{I}}(c), \ldots, {C_l}^{\mathcal{I}}(c)) \leqslant C^{\mathcal{I}}(c)$ whenever all ${C_i}^{\mathcal{I}}(c)$ are defined.

LP Component. $\mathcal{P}$ is a finite set of *vague rules* of the form (an example of a rule is shown in Example 4 below.)

$$R(\boldsymbol{x}) \leftarrow \exists \boldsymbol{y}.f(R_1(\boldsymbol{z}_1), \ldots, R_l(\boldsymbol{z}_l), p_1(\boldsymbol{z}'_1), \ldots, p_h(\boldsymbol{z}'_h)) \,,$$

where

1. R is an n-ary relation, every R_i is an n_i-ary relation,
2. $\boldsymbol{x}$ are the *distinguished variables*;
3. $\boldsymbol{y}$ are existentially quantified variables called the *non-distinguished variables*;
4. $\boldsymbol{z_i}, \boldsymbol{z'_j}$ are tuples of constants or variables in $\boldsymbol{x}$ or $\boldsymbol{y}$;
5. p_j is an n_j-ary *fuzzy predicate* assigning to each n_j-ary tuple $\boldsymbol{c}_j$ a *score* $p_j(\boldsymbol{c}_j) \in [0,1]_m$. Such predicates are called *expensive predicates* in [3] as the score is not pre-computed off-line, but is computed on query execution. We require that an n-ary fuzzy predicate p is *safe*, that is, there is not an m-ary fuzzy predicate p' such that $m < n$ and $p = p'$. Informally, all parameters are needed in the definition of p;
6. f is a *scoring* function $f \colon ([0,1]_m)^{l+h} \to [0,1]_m$, which combines the scores of the l relations $R_i(\boldsymbol{c}'_i)$ and the n fuzzy predicates $p_j(\boldsymbol{c}''_j)$ into an overall *score* to be assigned to the rule head $R(\boldsymbol{c})$. We assume that f is *monotone*, that is, for each $\boldsymbol{v}, \boldsymbol{v'} \in ([0,1]_m)^{l+h}$ such that $\boldsymbol{v} \leqslant \boldsymbol{v'}$, it holds $f(\boldsymbol{v}) \leqslant f(\boldsymbol{v'})$, where $(v_1, \ldots, v_{l+h}) \leqslant (v'_1, \ldots, v'_{l+h})$ iff $v_i \leqslant v'_i$ for all i. We also assume that the computational cost of f and all fuzzy predicates p_i is bounded by a constant.

We call $R(\boldsymbol{x})$ the *head* and $\exists \boldsymbol{y}.f(R_1(\boldsymbol{z}_1), \ldots, R_l(\boldsymbol{z}_l), p_1(\boldsymbol{z}'_1), \ldots, p_h(\boldsymbol{z}'_h))$ the *body* of the rule. We assume that relations occurring in $\mathcal{O}$ may appear in rules in $\mathcal{P}$ and that relations occurring in $\mathcal{F}$ do not occur in the head of rules and axioms (so, we do not allow that the fact relations occurring in $\mathcal{F}$ can be redefined by $\mathcal{P}$ or $\mathcal{O}$). As usual in deductive databases, the relations in $\mathcal{F}$ are called *extensional* relations, while the others are *intensional* relations.

Example 4. Consider again Example 2. The following rule may be used to retrieve a cheap single room in a hotel close to the city center:

$$q(x_1, x_2) \leftarrow \mathit{CloseHotelTable}(x_1, x_2, x_3, x_4, x_5) \cdot \mathit{cheap}(x_3) \,,$$

where

$$\mathit{cheap}(\mathit{price}) = \max(0, 1 - \frac{\mathit{price}}{250}) \,.$$

In the rule, *cheap* is a fuzzy predicate that computes the degree of cheapness of a given price. The overall score to be assigned to the retrieved hotels $\langle c_1, c_2 \rangle$ is computed as

the product (which is here the scoring function) of the degree of the closeness of a hotel (that is, the score of $CloseHotelTable(c_1, c_2, c_3, c_4, c_5)$) and the degree of cheapness of it ($cheap(c_3)$). Clearly, the product is a monotone score combination function. We will see that the instances of $q(x_1, x_2)$ together with their score will be

ID	$HOTEL$	s
1	*Verdi*	0.45
2	*Puccini*	0.26
3	*Rossini*	0.17 .

Semantically, an interpretation $\mathcal{I}$ is a *model* of a rule r of the form $R(\boldsymbol{x}) \leftarrow \exists \boldsymbol{y}.\phi(\boldsymbol{x}, \boldsymbol{y})$, where $\phi(\boldsymbol{x}, \boldsymbol{y}) = \exists \boldsymbol{y}.f(R_1(\boldsymbol{z}_1), \ldots, R_l(\boldsymbol{z}_l), p_1(\boldsymbol{z}'_1), \ldots, p_h(\boldsymbol{z}'_h))$, denoted $\mathcal{I} \models r$, iff for all $\boldsymbol{c} \in \Delta^n$ such that $R^{\mathcal{I}}(\boldsymbol{c})$ is defined, the following holds (where $\phi^{\mathcal{I}}(\boldsymbol{c}, \boldsymbol{c'})$ is obtained from $\phi(\boldsymbol{c}, \boldsymbol{c'})$ by replacing every R_i by $R_i^{\mathcal{I}}$ and every constant c by $c^{\mathcal{I}}$):

$$R^{\mathcal{I}}(\boldsymbol{c}) \geqslant \sup_{\boldsymbol{c'} \in \Delta \times \cdots \times \Delta,\ \phi^{\mathcal{I}}(\boldsymbol{c}, \boldsymbol{c'}) \text{ is defined}} \phi^{\mathcal{I}}(\boldsymbol{c}, \boldsymbol{c'}) \ .$$

We say $\mathcal{I}$ is a *model* of a knowledge base $\mathcal{K}$, denoted $\mathcal{I} \models \mathcal{K}$, iff $\mathcal{I}$ is a model of each expression $E \in \mathcal{F} \cup \mathcal{O} \cup \mathcal{P}$. We say $\mathcal{K}$ *entails* $R(\boldsymbol{c})$ to degree s, denoted $\mathcal{K} \models \langle R(\boldsymbol{c}), s \rangle$, iff for each model $\mathcal{I}$ of $\mathcal{K}$, it is true that $R^{\mathcal{I}}(\boldsymbol{c}) \geqslant s$ whenever $R^{\mathcal{I}}(\boldsymbol{c})$ is defined. The *greatest lower bound* of $R(\boldsymbol{c})$ relative to $\mathcal{K}$ is $glb(\mathcal{K}, R(\boldsymbol{c})) = \sup\{s \mid \mathcal{K} \models \langle R(\boldsymbol{c}), s \rangle\}$.

Example 5. The table in Example 4 reports the greatest lower bound of the instances of $q(x_1, x_2)$. In particular, $glb(\mathcal{K}, q(1, Verdi)) = 0.45$.

Example 6. Consider again Example 3. Now, suppose that in buying a car, preferably we would like to pay around \$12000 and the car should have less than 15000 km. Of course, our constraints on price and kilometers are not crisp as we may still accept to some degree, e.g., a car's cost of \$12200 and with 16000 km. Hence, these constraints are rather *vague*. We model this by means of so-called left-shoulder functions (see Fig. 4 for some typical fuzzy membership functions), which is a well known fuzzy membership function in fuzzy set theory. We may model the vague constraint on the cost with $ls(x; 10000, 14000)$ dictating that we are definitely satisfied if the price is less than \$10000, but can pay up to \$14000 to a lesser degree of satisfaction. Similarly, we may model the vague constraint on the kilometers with $ls(x; 13000, 17000)$.[1] We also set some preference (weights) on these two vague constraints, say the weight 0.7 to the price constraint and 0.3 to the kilometers constraint, indicating that we give more priority to the price rather than to the car's kilometers. The rules encoding the above conditions are represented in Fig. 5. Rule (1) in Fig. 5 encodes the preference on the price. Here, $ls(p; 10000, 14000)$ is the function that given a price p returns the degree of truth provided by the left-shoulder function $ls(\,\cdot\,; 10000, 14000)$ evaluated on the input p. Similarly, for rule (2). Rule (3) encodes the combination of the preferences by taking into account the weight given to each preference. The table below reports the instances of $Buy(x, p, k)$ together with their greatest lower bound.

[1] Recall that in our setting, all fuzzy membership functions provide a truth value in $[0, 1]_m$.

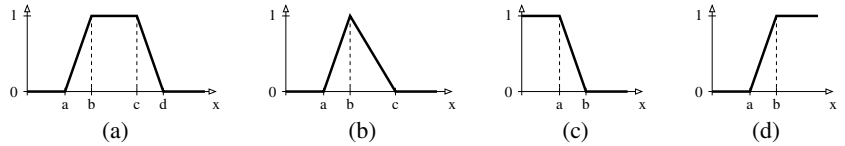

Fig. 4. (a) Trapezoidal function $trz(x; a, b, c, d)$, (b) triangular function $tri(x; a, b, c)$, (c) left shoulder function $ls(x; a, b)$, and (d) right shoulder function $rs(x; a, b)$

$$\begin{aligned} \mathit{Pref1}(x,p) &\leftarrow \min(\mathit{Cars}(x), \mathit{hasPrice}(x,p), \mathit{ls}(p; 10000, 14000)) \; ; & (1)\\ \mathit{Pref2}(x,k) &\leftarrow \min(\mathit{Cars}(x), \mathit{hasKM}(x,k), \mathit{ls}(k; 13000, 17000)) \; ; & (2)\\ \mathit{Buy}(x,p,k) &\leftarrow 0.7 \cdot \mathit{Pref1}(x,p) + 0.3 \cdot \mathit{Pref2}(x,k) \; . & (3) \end{aligned}$$

Fig. 5. The car buying rules

ID	*PRICE*	*KM*	s
455	*12500*	10000	0.56
34	*12000*	15000	0.50
1812	*11000*	16000	0.60 .

The basic inference problem that we are interested in here is the top-k retrieval problem, which is formulated as follows.

Top-k Retrieval. Given a knowledge base $\mathcal{K}$, retrieve k tuples $\langle \boldsymbol{c}, s\rangle$ that instantiate the query relation R with maximal scores (if k such tuples exist), and rank them in decreasing order relative to the score s, denoted

$$ans_k(\mathcal{K}, R) = \mathit{Top}_k\{\langle \boldsymbol{c}, s\rangle \mid s = \mathit{glb}(\mathcal{K}, R(\boldsymbol{c}))\} \; .$$

Example 7. It can be verified that the answer to the top-2 problem for Example 6 is

ID	*PRICE*	*KM*	s
1812	*11000*	16000	0.60
455	*12500*	10000	0.56 .

Whereas for Example 5 the answer to the top-2 problem is

ID	*HOTEL*	s
1	*Verdi*	0.45
2	*Puccini*	0.26 .

3 Top-k Query Answering for Deterministic KBs

We next provide a top-down top-k query answering algorithm.

We say that $\mathcal{K}$ is *deterministic* if for each relation symbol R there is at most one axiom or rule in $\mathcal{K}$ having R in its head. Given $\mathcal{K}$, we first note that we can remove the $\mathcal{O}$

component by transforming axioms into rules. Indeed, can rewrite $C_1 \sqcap \ldots \sqcap C_l \sqsubseteq C$ as $\sigma_C \leftarrow \min(\sigma_{C_1}, \ldots, \sigma_{C_l})$, where

$$\sigma_C = \begin{cases} A(x) & \text{if } C = A \\ R(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_n) & \text{if } C = \exists i{:}R\,, \end{cases}$$

and A is an atomic concept name. So, without loss of generality, we assume that $\mathcal{O}$ is empty. Obviously, we may also use a more expressive $\mathcal{O}$ component in which we consider axioms of the form $f(C_1, \ldots, C_l) \sqsubseteq C$ instead, where f is a score combination function. The translation (and so semantics) would be $\sigma_C \leftarrow f(\sigma_{C_1}, \ldots, \sigma_{C_l})$.

Concerning the computation of the top-k answers, of course, we always have the possibility to compute all answers, to rank them afterwards, and to select the top-k ones only. However, this requires computing the scores of all answers. We would like to avoid this in cases in which the extensional database is large and potentially too many tuples would satisfy the query.

A distinguishing feature of our query answering procedure is that we do not determine all answers, but collect, during the computation, answers incrementally together and we can stop as soon as we have gathered k answers above a computed threshold.

Overall, we build a procedure on top of current technology for top-k retrieval in databases, specifically on RankSQL [9]. In the database we store the facts and new derived facts and use RankSQL to retrieve incrementally new tuples. On top of it, we have a reasoning module, which deals with the rules of the KB.

The presentation of our algorithm proceeds as follows. We first present a top-k answering procedure for deterministic KBs. Then we address the more general case of non-deterministic KBs as well. In the following, given an intensional relation Q, we denote by r_Q the set of all rules $r : Q(\boldsymbol{x}) \leftarrow \phi \in \mathcal{P}$ (that is, the set of all rules r in $\mathcal{P}$ having Q in their head). Given $r : Q(\boldsymbol{x}) \leftarrow \phi \in \mathcal{P}$, we denote by $s(Q, r)$ the set of all *sons* of Q relative to r (that is, the set of all intensional relation symbols occurring in ϕ). We denote by $p(Q)$ the set of all *parents* of Q, that is, the set $p(Q) = \{R_i \colon Q \in s(R_i, r)\}$ (that is, the set of all relation symbols directly depending on Q).

The procedure $TopAnswers$ is detailed in Fig. 6. We assume that facts are stored into database tables, as specified in the facts component description. We also use some auxiliary functions and data structures: (i) for each intensional relation P, $rankedList(P)$ is a database relation containing the current top-ranked instances of P together with their score. For each P, the tuples $\langle \boldsymbol{c}, s \rangle$ in $rankedList(P)$ are ranked in decreasing order with respect to the score s. We do not allow $\langle \boldsymbol{c}, s \rangle$ and $\langle \boldsymbol{c}, s' \rangle$ to occur in $rankedList(P)$ with $s \neq s'$ (if so, then we remove the tuple with the lower score); (ii) the variable dg collects the relation symbols that the query relation Q depends on[2]; (iii) the array variable exp traces the rule bodies that have been "expanded" (the relation symbols occurring in the rule body are put into the active list); (iv) the variable in keeps track of the relation symbols that have been put into the active list so far due to an expansion (to avoid, to put the same relation symbol multiple times in the active list due to rule body expansion); (v) δ is a threshold with the property that if we have retrieved k tuples for Q

[2] Given a rule, the relation in the head *directly depends on* the relations in the body. *Depends on* is the transitive closure of the relation "directly depends on" with respect to a set of rules.

Procedure $TopAnswers(\mathcal{K}, Q, k)$
Input: KB $\mathcal{K}$, intensional query relation symbol Q, $k \geqslant 1$;
Output: Mapping $rankedList$ such that $rankedList(Q)$ contains top-k answers of Q
Init: $\delta = 1$, **for all** rules $r : P(\boldsymbol{x}) \leftarrow \phi$ in $\mathcal{P}$ **do**
if P intensional **then** $rankedList(P) = \emptyset$;
if P extensional **then** $rankedList(P) = T_P$ **endfor**

1. **loop**
2. $Active := \{Q\}$, $dg := \{Q\}$, $in := \emptyset$,
for all rules $r : P(\boldsymbol{x}) \leftarrow \phi$ **do** $exp(P, r) = false$;
3. **while** $Active \neq \emptyset$ **do**
4. **select** $P \in A$ where $r : P(\boldsymbol{x}) \leftarrow \phi$, $Active := Active \setminus \{P\}$, $dg := dg \cup s(P, r)$;
5. $\langle \boldsymbol{t}, s \rangle := getNextTuple(P, r)$
6. **if** $\langle \boldsymbol{t}, s \rangle \neq NULL$ **then** insert $\langle \boldsymbol{t}, s \rangle$ into $rankedList(P)$,
$Active := Active \cup (p(P) \cap dg)$;
7. **if not** $exp(P, r)$ **then** $exp(P, r) = true$,
$Active := Active \cup (s(P, r) \setminus in)$, $in := in \cup s(p, r)$;
endwhile
8. Update threshold δ;
9. **until** $rankedList(Q)$ does contain k top-ranked tuples with score above δ
or $rL' = rankedList$;
10. **return** top-k ranked tuples in $rankedList(Q)$;

Procedure $getNextTuple(P, r)$
Input: intensional relation symbol P and rule r: $P(\boldsymbol{x}) \leftarrow \exists \boldsymbol{y}.f(R_1(\boldsymbol{z}_1), \ldots, R_n(\boldsymbol{z}_l)) \in \mathcal{P}$;
Output: Next tuple satisfying the body of the r together with the score
loop
1. Generate next new instance tuple $\langle \boldsymbol{t}, s \rangle$ of P,
using tuples in $rankedList(R_i)$ and RankSQL
2. **if** there is no $\langle \boldsymbol{t}, s' \rangle \in rankedList(P, r)$ with $s \leqslant s'$ **then** exit loop
until no new valid join tuple can be generated
3. **return** $\langle \boldsymbol{t}, s \rangle$ if it exists **else return** $NULL$

Fig. 6. The top-k query answering procedure

with the score above δ, then we can stop, as it is guaranteed that any new tuple being an instance of Q has a score below δ. The threshold δ is determined by the RankSQL system and is described in more detail in [3,6].

Overall, the procedure works as follows. Assume, we are interested in determining the top-k answers of $Q(\boldsymbol{x})$. We start with putting the relation symbol Q in the *active* list of relation symbols $Active$. At each iteration step, we select a new relation symbol P from the queue $Active$ and get a new tuple ($getNexTuple(P, r)$) satisfying the rule body r whose head contains P with respect to the answers gathered so far. If the evaluation leads to a new answer for P ($\langle \boldsymbol{t}, s \rangle \neq NULL$), we update the current answer set $rankedList(P)$ and add all relations P_j directly depending on p to the queue $Active$. At some point, the active list will become empty and we have actually found correct answers of $Q(\boldsymbol{x})$. A threshold will be used to determine when we can stop

retrieving tuples. Indeed, the threshold determines when any newly retrieved tuple for Q scores lower than the current top-k, and thus cannot modify the top-k ranking (step 9). So, step 1 loops until we do not have k answers above the threshold, or two successive loops do not modify the current set of answers (step 9). Step 2 initializes the active list of relations. Step 3 loops until no relation symbol has to be processed anymore. In step 4, we select a relation symbol to be processed. In step 5, we retrieve the next answer for P. If a new answer has been retrieved (step 6, $\langle \boldsymbol{t}, s\rangle \neq NULL$), then we update the current answer set $rankedList(P)$ and add all relations P_j that directly depend on P to the queue $Active$. In step 7, we put once all intensional relation symbols appearing in the rule body of P in the active list for further processing.

Finally, the $getNextTuple$ procedure's (see Fig. 6) main purpose is, given a relation symbol P and a rule $r : P(\boldsymbol{x}) \leftarrow \phi$, to get back the next tuple (and its score) satisfying the conditions of the rule r. It essentially converts r into a RankSQL query to be submitted to the database engine (the translation is pretty standard) and returns the next top ranked unseen tuple.

Example 8. For instance, consider

$$Buy(x, p, k) \leftarrow 0.7 \cdot \mathit{Pref1}(x, p) + 0.3 \cdot \mathit{Pref2}(x, k)$$

as described in Example 6. Step 2 of the $getNextTuple$ procedure can be implemented as the RankSQL query

SELECT	p1.id, p1.price, p2.km
FROM	rankedList(Pref1) p1, rankedList(Pref2) p2
WHERE	p1.id = p2.id AND (exclude already processed tuples for rule r)
ORDER BY	0.7*p1.s + 0.3*p2.s
NEXT	1

The NEXT k statement allows incrementally to access to the next k tuples of the query (see the iMPro algorithm in [3]). We also use the condition "exclude already processed tuples for rule r" to guarantee that we do not join tuples twice for the rule r. This can be implemented using additional flags in $rankedList(Prefi)$. Afterwards, we can proceed with the other steps of the $getNextTuple$ procedure.

For more details on RankSQL, the interested reader may refer to [9,8]. Using RankSQL [9] has a further advantage as it directly provides us the threshold δ (see [3,6]) to be used to stop the computation.

Example 9. Assume that we have the following query rule

$$Q(x) \leftarrow \min(R_1(x, y), R_2(y, z)) \,,$$

where Q is the query relation, and R_1 and R_2 are extensional relations with tables as described in Fig. 7, left side. A top-2 retrieval computation is reported in Fig. 7, right side. In this case, the threshold δ is computed as (see [6])

$$\begin{aligned} \delta_1 &= \boldsymbol{t}_1^{\perp}.score \cdot \boldsymbol{t}_2^{\top}.score \\ \delta_2 &= \boldsymbol{t}_1^{\top}.score \cdot \boldsymbol{t}_2^{\perp}.score \\ \delta &= \max(\delta_1, \delta_2) \,, \end{aligned}$$

ID	R_1	R_2
1	$a\ b\ 1.0$	$m\ h\ 0.95$
2	$c\ d\ 0.9$	$m\ j\ 0.85$
3	$e\ f\ 0.8$	$f\ k\ 0.75$
4	$l\ m\ 0.7$	$m\ n\ 0.65$
5	$o\ p\ 0.6$	$p\ q\ 0.55$
$\vdots$	$\vdots\ \vdots\ \vdots$	$\vdots\ \vdots\ \vdots$

$TopAnswers$					
$It.$	$Active$	P	Δ_r	$rankedList(P)$	δ
1.	Q	Q	$\langle e, k, 0.75\rangle$	$\langle e, k, 0.75\rangle$	0.8
2.	Q	Q	$\langle l, h, 0.7\rangle$	$\langle e, k, 0.75\rangle, \langle l, h, 0.7\rangle$	0.75
3.	Q	Q	$\langle l, j, 0.7\rangle$	$\langle e, k, 0.75\rangle, \langle l, h, 0.7\rangle, \langle l, j, 0.7\rangle$	0.75
4.	Q	Q	$\langle l, n, 0.65\rangle$	$\langle e, k, 0.75\rangle, \langle l, h, 0.7\rangle,$ $\langle l, j, 0.7\rangle, \langle l, n, 0.65\rangle$	0.7

Fig. 7. Facts and computation of Example 9

$TopAnswers$					
$It.$	$Active$	P	Δ_r	$rankedList(P)$	δ
1.	Q	Q	$\langle 1, \mathit{Verdi}, 0.45\rangle$	$\langle 1, \mathit{Verdi}, 0.45\rangle$	–
2.	Q	Q	$\langle 2, \mathit{Puccini}, 0.26\rangle$	$\langle 1, \mathit{Verdi}, 0.45\rangle, \langle 2, \mathit{Puccini}, 0.26\rangle$	0.25

Fig. 8. Facts and computation of Example 10

where $\boldsymbol{t}_i^{\perp}$ is the last tuple seen in R_i, while $\boldsymbol{t}_i^{\top}$ is the top ranked one in R_i. With $\boldsymbol{t}_i.score$ we indicate the tuple's score.

The first call of $getNextTuple(Q)$ finds the tuple $\langle e, k, 0.75\rangle$. In the second call, we get $\langle l, h, 0.7\rangle$. In the third call, we get $\langle l, j, 0.7\rangle$. Finally, in the fourth call, we retrieve $\langle l, n, 0.65\rangle$. As now $rankedList(Q)$ contains two answers not smaller than the threshold 0.7, RankSQL will stop in providing new tuples in the $getNextTuple$ procedure, and we return $\{\langle e, k, 0.75\rangle, \langle l, h, 0.7\rangle\}$. Note that no additional retrieved answer may have a score above 0.7. Indeed, the next one would be $\langle o, q, 0.55\rangle$. Hence, in this case, not all tuples are processed.

Example 10. Consider Example 4 and the query

$$q(x_1, x_2) \leftarrow \mathit{CloseHotelTable}(x_1, x_2, x_3, x_4, x_5) \cdot \mathit{cheap}(x_3)\ .$$

The top-2 computation is shown in Fig. 8. After the second iteration, RankSQL stops (see [3]) in providing new tuples in the $getNextTuple$ as we have already two tuples for q above the threshold, and RankSQL also knows that any successively retrieved tuple have a score below $0.25 \cdot 1.0 = 0.25$ (1.0 is assumed as the highest possible score for the degree of cheapness of the price for the single room in hotel Rossini). Of course, in the worst case, RankSQL has to sequentially scan the whole table as we may not know in advance how the fuzzy predicate cheap evaluates on the tuples. In fact, the tuple with the lowest degree of closeness may end up with the highest degree of cheapness.

Let us briefly discuss the computational complexity. Let $\mathcal{K}$ be a KB. As we have seen, the $\mathcal{O}$ component can be translated into $\mathcal{P}$, and thus $\mathcal{O} = \emptyset$ can be assumed. Let D_Q be the set of relation symbols that *depend* on the query relation symbol Q. Of course, only relation symbols in D_Q may appear in $Active$. Let $ground(\mathcal{F} \cup \mathcal{P})$ denote the grounding of the logic program $\mathcal{F} \cup \mathcal{P}$. For a rule r_i of the form $R_i(\boldsymbol{x}) \leftarrow \exists \boldsymbol{y}.\phi_i(\boldsymbol{x}, \boldsymbol{y})$, let k_i be the arity of the body of the rule r, and let k be the maximal arity for all rules

in $\mathcal{P}$. Let p_i be of arity n_i, and let n be the maximal arity of relation symbols. Therefore, the number of ground instances of this rule is bounded by $|HU|^{k_i}$, and thus is bounded by $|HU|^k$, where HU is the Herbrand universe of $\mathcal{F} \cup \mathcal{P}$. Similarly, the number of ground instance of R_i is bounded by $|HU|^{n_i}$, and thus is bounded by $|HU|^n$. Let c_i be the cost of evaluating the score of the rule body of r_i.

Now, observe that any tuple's score is increasing as it enters in $rankedList$. Hence, the truth of any ground instance of relation symbol R_i is increasing as R_i enters in the $Active$ list (step 6), except it enters due to step 7, which may happen one time only. Therefore, each R_i will appear in $Active$ at most $O(|HU|^{n_i} \cdot m + 1)$ times,[3] as a relation symbol is only re-entered into $Active$ if a ground instance of $R_i(\boldsymbol{x})$ evaluates to an increased value, plus the additional entry due to step 7. As a consequence, the worst-case complexity is

$$O(\sum_{R_i \in D_Q} c_i \cdot (|HU|^{n_i} \cdot m + 1)) .$$

If for ease we assume that the cost c_i is $O(1)$ and that m is a fixed parameter (that is, a constant), then we get a worst-case complexity of

$$O(|ground(\mathcal{F} \cup \mathcal{P})|) ,$$

and so $TopAnswer$ is exponential with respect to $|\mathcal{F} \cup \mathcal{P}|$ (combined complexity), but polynomial in $|\mathcal{F}|$ (data complexity), and so is as for classical Datalog. The complexity result is not surprising as we resemble a top-down query answering procedure for Datalog. We have the following termination and correctness result (termination is guaranteed by the finiteness of $[0, 1]_m$ and the monotonicity of the score combination functions).

Theorem 1. *Let $\mathcal{K}$ be a deterministic KB, and let Q be a query. Then, $TopAnswers(\mathcal{K}, Q, k)$ terminates with $TopAnswers(\mathcal{K}, Q, k) = ans_k(\mathcal{K}, Q)$.*

4 Top-k Query Answering for General KBs

Our top-k retrieval algorithm is based on the fact that whenever we find k tuples with score above the threshold we can stop. Unfortunately, if R is in the head of more than one rule, this is no longer true, and we have to modify slightly the computation of the threshold. Note that since interpretations involve partial functions, e.g., $R(x) \leftarrow R_1(x)$ and $R(x) \leftarrow R_2(x)$ are not the same as $Q(x) \leftarrow \max(R_1(x), R_2(x))$ (in the latter case, x has to belong to both R_1 and R_2, while in the former case, it suffices that x belongs either to R_1 or to R_2).[4]

To accommodate the case where not always $|r_R| \leqslant 1$, we have to make the following modifications to the $TopAnswer$ procedure as shown in Fig. 9. Essentially, we process all rules related to a selected relation symbol P (step 4.1), and the threshold is updated according to step 8, where each δ_r, for $r \in r_Q$, is the threshold determined by RankSQL

[3] We recall that m is the parameter in $[0, 1]_m$.

[4] If for an application we may live with $Q(x) \leftarrow \max(R_1(x), R_2(x))$, then this guarantees that we may restrict our attention to deterministic KBs.

Procedure $TopAnswersGen(\mathcal{K}, Q, k)$
Input: KB $\mathcal{K}$, intensional query relation Q, $k \geqslant 1$;
Output: Mapping $rankedList$ such that $rankedList(Q)$ contains top-k answers of Q
Init: $\delta = 1$, $\forall r \in r_Q.\delta_r = 1$, **for all** rules $r : P(\boldsymbol{x}) \leftarrow \phi$ in $\mathcal{P}$ **do**
if P intensional **then** $rankedList(P) = \emptyset$;
if P extensional **then** $rankedList(P) = T_P$ **endfor**
1. **loop**
2. $Active := \{Q\}$, $dg := \{Q\}$, $in := \emptyset$,
for all rules $r : P(\boldsymbol{x}) \leftarrow \phi$ **do** $exp(P, r) = false$;
3. **while** $Active \neq \emptyset$ **do**
4. **select** $P \in A$, $Active := Active \setminus \{P\}$;
4.1 **for all** $r : P(\boldsymbol{x}) \leftarrow \phi \in \mathcal{P}$, $dg := dg \cup s(P, r)$;
5. $\langle \boldsymbol{t}, s\rangle := getNextTuple(P, r)$
6. **if** $\langle \boldsymbol{t}, s\rangle \neq NULL$ **then** insert $\langle \boldsymbol{t}, s\rangle$ into $rankedList(P)$,
$Active := Active \cup (p(P) \cap dg)$;
7. **if not** $exp(P, r)$ **then** $exp(P, r) = true$,
$Active := Active \cup (s(P, r) \setminus in)$, $in := in \cup s(p, r)$;
endfor
endwhile
8. Update threshold as $\delta = \max_{r \in r_Q} \delta_r$;
9. **until** $rankedList(Q)$ does contain k top-ranked tuples with score above δ
or $rL' = rankedList$;
10. **return** top-k ranked tuples in $rankedList(Q)$;

Fig. 9. The top-k query answering procedure for general KBs

as for the deterministic case. Here, we have to use $\max$, since a tuple instantiating the query relation Q may be derived for *some* of the rules in r_Q. The following example illustrates the basic principle behind the $TopAnswersGen$ procedure.

Example 11. Consider the rules

$$r_1 : Q(x) \leftarrow R_1(x)\,;$$
$$r_2 : Q(x) \leftarrow P(x)\,;$$
$$r_3 : P(x) \leftarrow R_2(x)$$

and facts in Fig. 10, left side. The top-2 computation is shown in Fig. 10, right side. After step 5, we can stop, since we already have two answers with a score above the threshold 0.4, and we do not need to continue anymore. Note that any successively retrieved answer, e.g., $\langle e, 0.3\rangle$ has a score below the threshold 0.4.

The complexity is as for the deterministic case, and we have the following termination and correctness result.

Theorem 2. *Let $\mathcal{K}$ be a general KB, and let Q be a query. Then, $TopAnswersGen(\mathcal{K}, Q, k)$ terminates with $TopAnswersGen(\mathcal{K}, Q, k) = ans_k(\mathcal{K}, Q)$.*

R_1	
ID	s
a	0.5
b	0.4
e	0.3
f	0.1

R_2	
ID	s
c	0.7
d	0.2
g	0.1
h	0.05

$TopAnswers$							
$It.$	$Active$	P	Δ_r	$rankedList(P)$	δ	δ_{r_1}	δ_{r_2}
1.	Q	Q	$\langle a, 0.5\rangle$	$\langle a, 0.5\rangle$	1.0	0.5	1.0
2.	P	P	$\langle c, 0.7\rangle$	$\langle c, 0.7\rangle$	–	–	–
3.	Q	Q	$\langle b, 0.4\rangle$	$\langle a, 0.5\rangle, \langle b, 0.4\rangle$	1.0	0.4	1.0
	Q	Q	$\langle c, 0.7\rangle$	$\langle c, 0.7\rangle, \langle a, 0.5\rangle, \langle b, 0.4\rangle$	0.7	0.4	0.7
4.	P	P	$\langle d, 0.2\rangle$	$\langle c, 0.7\rangle, \langle d, 0.2\rangle$	–	–	–
5.	Q	Q	$\langle d, 0.2\rangle$	$\langle c, 0.7\rangle, \langle a, 0.5\rangle, \langle b, 0.4\rangle, \langle d, 0.2\rangle$	0.4	0.4	0.2

Fig. 10. Facts and computation of Example 11

5 Related Work

While there are many works addressing the top-k problem for vague queries in databases (cf. [1,3,5,4,6,7,9,8,10]), little is known for the corresponding problem in knowledge representation and reasoning. For instance, [21] considers non-recursive fuzzy logic programs in which the score combination function is a function of the score of the atoms in the body only (no expensive fuzzy predicates are allowed). The work [16] considers non-recursive fuzzy logic programs as well, though the score combination function may consider expensive fuzzy predicates. However, a score combination function is allowed in the query rule only. We point out that in the case of non-recursive rules and/or axioms, we may rely on a query rewriting mechanism, which, given an initial query, rewrites it, using rules and/or axioms of the KB, into a set of new queries until no new query rule can be derived (this phase may require exponential time relative to the size of the KB). The obtained queries may then be submitted directly to a top-k retrieval database engine. The answers to each query are then merged using the disjunctive threshold algorithm given in [16]. The works [17,15] address the top-k retrieval problem for the description logic *DL-Lite* only, though recursion is allowed among the axioms. Again, the score combination function may consider expensive fuzzy predicates. However, a score combination function is allowed in the query only. The work [19] shows an application of top-k retrieval to the case of multimedia information retrieval by relying on a fuzzy variant of *DLR-Lite*. Finally, [18] addresses the top-k retrieval for general (recursive) fuzzy LPs, though no expensive fuzzy predicates are allowed. Closest to our work is clearly [18]. In fact, our work extends [18] by allowing expensive fuzzy predicates, which have the effect that the threshold mechanism designed in [18] does not work anymore. Furthermore, in this paper, we made an effort to plug-in current top-k database technology, while [18] does not and provides an ad-hoc solution. Though we have not yet performed experimental evaluations, we hope that this choice, beside allowing a more expressive language, will provide better efficiency.

6 Summary and Outlook

The top-k retrieval problem is an important problem in logic-based languages for the Semantic Web. We have addressed this issue for a combination of a fuzzy DL and LP.

An implementation of our algorithm is under development, by relying on RankSQL. Other main topics for future work include: (i) Can we apply similar ideas to more expressive DLs and/or non-monotonic LPs? (ii) How can we approach the top-k problem under a probabilistic setting, or more generally under uncertainty, possibly relying on emerging top-k retrieval systems for uncertain database management [13,14]?

Acknowledgments. Thomas Lukasiewicz is supported by the German Research Foundation (DFG) under the Heisenberg Programme. We thank the reviewers for their constructive and useful comments, which helped to improve this work.

References

1. Bruno, N., Chaudhuri, S., Gravano, L.: Top-k selection queries over relational databases: Mapping strategies and performance evaluation. ACM TODS 27(2), 153–187 (2002)
2. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. KR-2006, pp. 260–270 (2006)
3. Chang, K.C.-C., Hwang, S.-W.: Minimal probing: Supporting expensive predicates for top-k queries. In: Proc. SIGMOD-2002, pp. 346–357 (2002)
4. Fagin, R.: Combining fuzzy information: An overview. SIGMOD Rec. 31(2), 109–118 (2002)
5. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: Proc. PODS-2001 (2001)
6. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K.: Supporting top-k join queries in relational databases. In: Aberer, K., Koubarakis, M., Kalogeraki, V. (eds.) Databases, Information Systems, and Peer-to-Peer Computing. LNCS, vol. 2944, pp. 754–765. Springer, Heidelberg (2004)
7. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K., Elmongui, H.G., Shah, R., Vitter, J.S.: Adaptive rank-aware query optimization in relational databases. ACM TODS 31(4), 1257–1304 (2006)
8. Li, C., Chang, K.C.-C., Ilyas, I.F.: Supporting ad-hoc ranking aggregates. In: Proc. SIGMOD-2006, pp. 61–72 (2006)
9. Li, C., Chang, K.C.-C., Ilyas, I.F., Song, S.: RankSQL: Query algebra and optimization for relational top-k queries. In: Proc. SIGMOD-2005, pp. 131–142 (2005)
10. Marian, A., Bruno, N., Gravano, L.: Evaluating top-k queries over web-accessible databases. ACM TODS 29(2), 319–362 (2004)
11. Meghini, C., Sebastiani, F., Straccia, U.: A model of multimedia information retrieval. J. ACM 48(5), 909–970 (2001)
12. Ragone, A., Straccia, U., Di Noia, T., Di Sciascio, E., Donini, F.M.: Vague knowledge bases for matchmaking in P2P e-marketplaces. In: Proc. ESWC-2007, pp. 414–428 (2007)
13. Ré, C., Dalvi, N., Suciu, D.: Efficient top-k query evaluation on probabilistic data. In: Proc. ICDE-2007, pp. 886–895 (2007)
14. Soliman, M.A., Ilyas, I.F., Chang, K.C.: Top-k query processing in uncertain databases. In: Proc. ICDE-2007, pp. 896–905 (2007)
15. Straccia, U.: Answering vague queries in fuzzy *DL-Lite*. In: Proc. IPMU-2006, pp. 2238–2245 (2006)
16. Straccia, U.: Towards top-k query answering in deductive databases. In: Proc. SMC-2006, pp. 4873–4879 (2006)
17. Straccia, U.: Towards top-k query answering in description logics: The case of *DL-Lite*. In: Proc. JELIA-2006, pp. 439–451 (2006)
18. Straccia, U.: Towards vague query answering in logic programming for logic-based information retrieval. In: Proc. IFSA-2007, pp. 125–134 (2007)

19. Straccia, U., Visco, G.: DLMedia: An ontology mediated multimedia information retrieval system. In: Proc. DL-2007 (2007)
20. Vojtáš, P.: Fuzzy logic programming. Fuzzy Sets and Systems 124, 361–370 (2001)
21. Vojtáš, P.: Fuzzy logic aggregation for semantic web search for the best (top-k) answer. In: Sanchez, E. (ed.) Fuzzy Logic and the Semantic Web. Capturing Intelligence, ch. 17, pp. 341–359. Elsevier, Amsterdam (2006)