FUKG: Answering Flexible Queries over Knowledge Graphs

José Félix Yagüe[1], Ignacio Huitzil[2], Carlos Bobed[13], and Fernando Bobillo[13]

[1] University of Zaragoza, Zaragoza, Spain
[2] IKR3 Lab, University of Milano-Bicocca, Milano, Italy
[3] Aragon Institute of Engineering Research (I3A), Zaragoza, Spain
{cbobed,fbobillo}@unizar.es

Abstract.

Purpose. The increasing interest in Knowledge Graphs to represent real-world knowledge and the common need to manage imprecise knowledge in many real-world applications demand the study of approaches to solve flexible queries over Knowledge Graphs.
Design/methodology/approach. By introducing Fuzzy Logic in the query answering process, we are able to obtain a novel algorithm to solve flexible queries over Knowledge Graphs. Our approach is implemented in the FUKG system, a software tool with an intuitive user graphical interface.
Findings. Our approach makes it possible to reuse Semantic Web standards (RDF, SPARQL, and OWL 2) and builds a fuzzy layer on top of them. The application to a use case shows that the system can aggregate information in different ways by selecting different fusion operators, adapting to different user needs.
Originality. Our approach is more general than similar previous works in the literature and provides a specific way to represent the flexible restrictions (using Fuzzy OWL 2 datatypes).

Keywords: Knowledge Graphs, Fuzzy Logic, Flexible Querying
Article classification: Article, Research paper.

1 Introduction

Knowledge is essential in Artificial Intelligence applications, which need to represent a domain and reason with it in a proper way. In this century, the most popular options to represent the knowledge in many domains and applications are Semantic Web technologies. Such technologies include ontologies, or formal and shared specifications of the vocabulary of a domain of interest Staab and Studer (2004), Knowledge Graphs Hogan et al. (2021), a graph-based model to capture data at large scale, and Linked Data, a set of best practices for publishing and connecting data on the Web Bizer et al. (2009). In a typical situation, there is a Knowledge Graph representing the relevant knowledge in RDF language, as a set of subject-predicate-object triples, where some of the nodes of the Knowledge Graph are described using an OWL ontology, and there are some relationships linking the nodes of the local knowledge graph to other external nodes.

Another important problem to address in intelligent applications is managing different types of uncertainty. Among the many existing types of imperfect knowledge, we are interested in vagueness and imprecision. They happen very often in many real-world applications for several reasons, such as the natural vagueness of natural language or the intrinsic imprecision of source data (e.g. sensors or machine learning approaches). To represent such kind of knowledge and reason with it, Fuzzy Set Theory and Fuzzy Logic have been applied to many successful use cases Zadeh (1965). Fuzzy sets allow to represent the partial membership of an element to a set, and Fuzzy Logic allows to manage propositions which are partially true and make deductions through approximate reasoning.

In order to support imprecise knowledge, the idea of extending different models with ideas from Fuzzy Logic seems very natural, and there are examples of fuzzy databases, fuzzy neural networks, fuzzy modeling languages, etc. Klir and Yuan (1995). Semantic Web technologies are not an exception, and many fuzzy extensions have been proposed, but the literature has mainly focused on fuzzy ontologies Lukasiewicz and Straccia (2008), Zhang et al. (2016) or fuzzy Description Logics Bobillo et al. (2015), as the main formalism behind fuzzy ontologies. Unfortunately, the combination of Fuzzy Logic and Knowledge Graphs has not received a similar attention, although it would be crucial to support flexible queries, including imprecise terms such as "cheap", "fast", or "recent" when, for example, looking for means of transportation.

In the field of fuzzy ontologies, a family of reasoning algorithms (crisp representation algorithms) is based on a reduction to classical ontologies Bobillo et al. (2012), Bobillo (2016). This makes it possible to reuse classical semantic reasoners (such as Pellet Sirin et al. (2007)) and other existing resources. More recently, minimalist reasoning algorithms solve some flexible queries (restricted from an expressivity point of view) over classical ontologies by reusing classical semantic reasoners and building a fuzzy layer on top of them as a series of additional steps Huitzil et al. (2020, 2021).

In this paper, we will propose a novel approach to answer flexible queries over classical Knowledge Graphs. While previous work focuses on the support of fuzzy axioms (using non-standard RDF), we instead focus on fuzzy datatypes. In particular, our flexible queries will be described using fuzzy sets to constrain the values of the numerical data properties. Furthermore, inspired by minimalist reasoning algorithms for fuzzy ontologies, our approach is able to stick to the Semantic Web standards, using standard RDF and SPARQL query endpoints, and performing a series of additional steps on top of them to do the necessary computations to deal with the fuzzy part.

This paper is a revised and extended version of Yagüe et al. (2022) with the following main differences: a more general definition of the flexible queries (including both conjunctive and disjunctive interpretations, several hedges rather than a single one, and top-k results), a more general algorithm to solve the queries, a description of an implementation in the FUKG tool, and a discussion of a real use case involving DBpedia.

The remainder of this paper is structured as follows. In Section 2 we overview some essential background knowledge needed to follow the rest of the paper. Then, Section 3 discusses our novel approach to represent and answer flexible queries over Knowledge Graphs, while Section 4 discusses the implementation of a prototype. Section 5 illustrates the usefulness of our approach by addressing a practical use case.

Finally, Section 6 overviews some related work, and Section 7 summarizes the main conclusions and identifies some ideas for future research.

2 Background

This section will firstly provide some background on Knowledge Graphs (Section 2.1) and then on Fuzzy Logic (Section 2.2).

2.1 Knowledge Graphs

Knowledge Graphs have been there for quite a long time under different names (e.g., Semantic Networks), but they have lately received a lot of attention since Google adopted the term in 2012. There is not a unique definition of what a Knowledge Graph is, but they are most of the times viewed as labeled directed graphs, i.e., a Knowledge Graph is a labeled directed graph (E, R, L), where E are entities, R are relations between such entities, and L is a labeling function mapping each element in the graph to its name/type. Influenced indeed by the RDF data model, this definition derives into regarding them as sets of triples subject-predicate-object (SPO) triples. However, the general notion of Knowledge Graph is broader than that. Without binding the definition to any particular data model, Hogan et al. Hogan et al. (2021) adopt the following definition: *a Knowledge Graph is a graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent relations between these entities*[4]. Note how this definition emphasizes the identity dimension of the nodes in the graph: each node must represent an entity of interest. In fact, we can find different levels of abstraction depending on how identity is dealt with and the kind of entities to be represented. Table 1 contains a comparison depending of these aspects, including (lightweight) KG and ontologies.

Table 1: Comparison among lightweight knowledge graphs, knowledge graphs, and ontologies

Feature	**Lightweight KGs**	**KGs**	**Ontologies**
Identity	String comparison	URI	URI
Entities	Instances	Instances	Instances, concepts, and properties
Schema	None	Inexpressive	Expressive
Reasoning	None	Limited	Powerful

The lower expressivity level would correspond to Lightweight KGs, where the nodes and relationships are just tagged with raw strings and the identity matching would be reduced to string matching. An example of these KGs could be the result of an Open Information Extraction procedure, where the main goal is to shallow parse text and

[4] The interested reader can find pointers to alternative definitions in Hogan et al. (2021).

structure it using triples, e.g., ("Obama", "was born in", "Honolulu"). When we move to a shared representation with global identifiers (e.g., URIs), we would be considering the KGs of the above definition. Following with the previous example, we could consider the DBpedia Lehmann et al. (2015), and see that the same triple can be stated using a shared vocabulary $\langle dbr:Barack\_Obama,\ dbo:birthPlace,\ dbr:Honolulu\rangle$, where instead of representing the elements just with the strings we have found in the text, we are using URIs that are unique globally for the entities we are describing. Finally, KGs usually come along with ontologies, which in turn are KGs themselves but they contain schema knowledge to share and represent the domain that the KG is about. In the example, the ontology could contain URIs for concepts such as `Person` or `Place` to state that $\langle dbr:Barack\_Obama,\ isA,\ dbo:Person\rangle$.

The previous classification and definition of KGs is heavily influenced by RDF graphs, which are labeled directed graphs. RDF is the W3C standard language for representing information in the Web Schreiber and Raimond (2014). It is based on triples of the form $\langle s,p,o\rangle$, where s is the subject, p is the property, and o is the object (or value), stating that s is related to o via the property p. In general, such relationships are not symmetrical.

To define the schema that the RDF graph follows, different languages with different expressivities can be used, ranging from RDF-Schema, also called RDF-S Brickley and Guha (2014), to OWL Cuenca-Grau et al. (2008) and their profiles. While using RDF-S one can only define hierarchies of concepts and properties, domain and ranges of properties, and the classes that individuals belongs to, OWL-2 provides a much richer language to model expressive ontologies in different fragments of Description Logics Baader et al. (2003).

SPARQL is RDF standard query language Pérez et al. (2006). It supports four different types of queries (namely, SELECT, ASK, CONSTRUCT, and DESCRIBE) whose body is described in terms of basic graph patterns (BGPs) composed using free variables. Such BGPs are to be matched crisply to the underlying graph in order to provide possible mappings that fulfill the query conditions.

2.2 Fuzzy Logic

Fuzzy Logic is widely used to manage imprecise and vague knowledge. It is a generalization of classical logic proposed by Zadeh where statements are not necessarily either true or false, but hold to some degree of truth Zadeh (1965).

The cornerstone of Fuzzy Logic is the concept of fuzzy set, which is a generalization of a classical set where elements can have a partial membership. A fuzzy set A is characterized by a membership function $\mu_A(x)$ which associates with each object x a real number in [0,1] representing the membership degree of x in A. As in classical sets, 0 means no-membership and 1 full membership, but now an intermediate value between 0 and 1 denotes partial membership to F.

To specify fuzzy membership functions, common options are the trapezoidal (Figure 1 (a)), triangular (Figure 1 (b)), left-shoulder (Figure 1 (c)), right-shoulder (Figure 1 (d)), and linear (Figure 1 (e) membership functions.

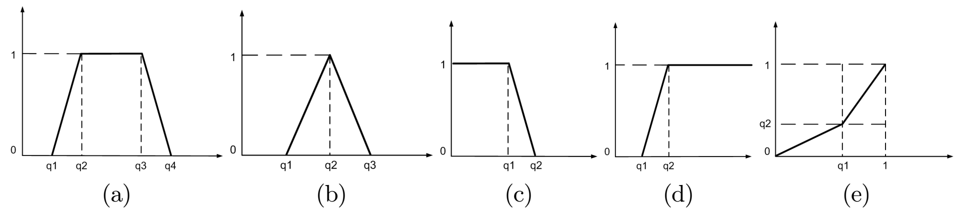


Fig. 1: (a) Trapezoidal; (b) Triangular; (c) Left-shoulder; (d) Right shoulder; (e) Linear fuzzy membership functions

Example 1. In the field of means of transportation, a fuzzy set HighmaxSpeedKmh can be defined using a a triangular function **triangular**(500,1000,1500). If the maximum speed of dbr:Fiat_G.91Y is 1110 km/h, its membership degree to HighmaxSpeedKmh is: $\mu_{\mathsf{HighmaxSpeedKmh}}(\mathsf{dbr{:}Fiat\_G.91Y}) = (\textbf{triangular}(500{,}1000{,}1500))(1110) = 0.8$. □

Fuzzy Logic enables approximate reasoning. Logical operations over classical sets are also generalized to the fuzzy case. To compute the conjunction, disjunction, complement and implication over fuzzy sets one can use different families of functions, namely a *t-norm* function $\otimes$, a *t-conorm* function $\oplus$, a *negation* function $\ominus$ and an *implication* function $\Rightarrow$ (see Klir and Yuan (1995) for details). For example, the minimum and the product are t-norms, while the maximum and the probabilistic sum are t-conorms.

In addition to logical operators, other ways to combine fuzzy sets are possible. An aggregation operator is a function that takes n values in $[0{,}1](x_1{,}x_2{,}...{,}x_n)$ and returns a single value in [0,1]. Some examples are the weighted mean (WMEAN) or the *Ordered Weighted Averaging* (OWA) operator.

– Given a vector of weights $[w_1, w_2, \ldots, w_n]$ such that $\forall i \in \{1, \ldots, n\}, w_i \in [0,1]$ and $\sum_{i=1}^{n} w_i = 1$, WMEAN is defined as $\sum_{i=1}^{n} w_i x_i$. That is, the i-th weight corresponds to the i-th value to be aggregated.
– On the other hand, OWA is defined as $\sum_{i=1}^{K} w_i x_{\sigma(i)}$, where $\sigma(i)$ is a permutation such that $x_{\sigma(1)} \geq x_{\sigma(2)} \geq \cdots \geq x_{\sigma(K)}$ Yager (1988). That is, the i-th weight corresponds to the i-th largest value to be aggregated.

Selecting the appropriate weights of OWA operators is not a trivial problem. One of the most popular solutions is using *quantifier-guided aggregation* Yager (1996). Given a Regular Increasing Monotone (RIM) fuzzy quantifier $Q:[0{,}1] \rightarrow [0{,}1]$, which satisfies the boundary conditions $Q(0)=0$ and $Q(1)=1$, and is monotone increasing, the weights of an OWA operator of dimension n can be defined as $w_i = Q(\frac{i}{n}) - Q(\frac{i-1}{n})$, $\forall i \in \{1{,}...{,}n\}$. Right-shoulder and linear functions can be used to describe RIMs.

Example 2. **right**(1/3,2/3) is a RIM fuzzy quantifier. For $n=3$, it leads to the vector of weights [0,1,0] since:

- $w_1 = Q(\frac{1}{3}) - Q(0) = 0 - 0 = 0$
- $w_2 = Q(\frac{2}{3}) - Q(\frac{1}{3}) = 1 - 0 = 1$

- $w_3 = Q(1) - Q(\frac{2}{3}) = 1 - 1 = 0$

Using this vector of weights, OWA discards the highest and the lowest value to be aggregated, as they have a 0 weight, and returns the middle value. □

Another important element of fuzzy logic are *fuzzy hedges* (also called fuzzy modifiers), which modify the shape of a fuzzy set by altering its membership function. The most popular examples are the weakening modifier very, characterized by the function $\mathsf{very}(x) = x^2$, and the increasing modifier few, defined as $\mathsf{few}(x) = \sqrt{x}$.

Example 3. By applying very to the fuzzy set HighmaxSpeedKmh defined in Example 1, we get the fuzzy set VeryHighmaxSpeedKmh. For example, for dbr:Fiat_G.91Y, $\mu_{\mathsf{VeryHighmaxSpeedKmh}}(1100) = \mathsf{very}(\mu_{\mathsf{highmaxSpeedKmh}}(1100)) = (\mu_{\mathsf{highmaxSpeedKmh}}(1100))^2 = 0.8^2 = 0.64$. Note that as the membership degree to the modified fuzzy set is in [0,1], by squaring it, we reinforce the need to belong "very" to the set. □

3 Flexible queries

This section will firstly describe how to characterize flexible queries (Section 3.1) and then we will address how to solve them (Section 3.2).

3.1 Representing the queries

Given a Knowledge Graph $\mathcal{K}$, a flexible query can be characterized by the following parameters:

- A set of classes $C_1, C_2, ..., C_N$
- A set of functional numerical data properties $P_1, P_2, ..., P_n$
- A list of fuzzy datatypes $D_1, D_2, ..., D_n$
- A list of fuzzy hedges $H_1, H_2, ..., H_n$
- An optional retrieval operator for the classes $\diamond$
- An optional fusion operator for the properties $@ : [0,1]^n \to [0,1]$
- An optional maximum of results k

For the classes, there are two possible retrieval operators, namely the union $\vee$ and the intersection $\wedge$. For the properties, possible fusion operators include t-norms, t-conorms, or other aggregation operators (such as weighted mean, or OWA).

Example 4. Given a Knowledge Graph about means of transportation, a possible flexible query to retrieve "motorcycles or automobiles with very high maximum speed, high cruise speed, and neutral empty weight" might be described as:

- Classes: dbo:Motorcycle and dbo:Automobile
- Data properties: dbp:maxSpeedKmh, dbp:emptyWeightKg and dbp:cruiseSpeedKmh
- Fuzzy datatypes: HighmaxSpeedKmh, defined as **triangular**(500, 1000, 1500), NeutralemptyWeightKg, defined as **triangular**(1000,3000,5000), and HighcruiseSpeedKmh, defined as **triangular**(500,750,1000)

- Fuzzy hedges: very (for dbp:maxSpeedKmh) none (for dbp:emptyWeightKg), and none (for dbp:cruiseSpeedKmh).
- Retrieval operator for the classes: union $\vee$ (any of the classes)
- Fusion operator for the properties: product t-norm
- Number of results: 999 □

We propose to rely on Fuzzy OWL 2 datatypes Bobillo and Straccia (2011), which includes trapezoidal, triangular, left-shoulder, and right-shoulder datatypes. In Fuzzy OWL 2, a datatype declaration can be associated with an OWL 2 annotation encoding a fuzzy membership function, using an XML-like syntax. Interestingly, such annotations can be encoded as triples in the Knowledge Graph.

Example 5. To express that fuzzy datatype HighmaxSpeedKmh corresponds to a triangular function **triangular**(500,1000,1500), we use the following set of RDF triples:

```
@PREFIX ex: <http://www.example.org/beer/> .
@PREFIX owl: <http://www.w3.org/2002/07/owl#> .
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
ex:fuzzyLabel rdf:type owl:AnnotationProperty .
ex:HighmaxSpeedKmh rdf:type rdfs:Datatype .
ex:HighmaxSpeedKmh ex:fuzzyLabel """<fuzzyOwl2 fuzzyType=\"datatype\">
 <Datatype type=\"triangular\" a=\"500\" b=\"1000\" c=\"1500\" />
 </fuzzyOwl2>""" .
```

□

Because there are several optional parameters, it is important to define the default values. It seems clear that, if no maximum of results is specified, all non-zero results should be retrieved. However, the choice of the retrieval and fusion operators could depend on the implementation. As we will see in Section 4, when they are not specified we propose *(i)* to retrieve instances of all classes ($\wedge$) and *(ii)* to combine values using the product.

3.2 Solving flexible queries

To solve a flexible query, we propose Algorithm 1. Firstly, we initialize the set of responses QR. Line 2 retrieves the instances belonging to all or to some of the classes, according to the operator $\diamond$, and then Line 3 retrieves the fuzzy datatype definitions F_i for each fuzzy datatype D_i in the query. Next, we compute for each instance of the classes the degree of satisfaction to the query. For each property P_i, Lines 5–12 compute the membership degree of the value y_i of P_i for individual x, to the fuzzy datatype F_i, and then a fuzzy hedge H_i is applied. It is worth to note that if the value of some property P_i is not defined (so getPropertyValue returns a NULL value), the value r_i^x is not computed for that property. Next, the algorithm checks that individual x has a value for all the properties in the query, so the cardinalities of r_i^x and P_i coincide. If this is the case, Line 14 combines all the membership degrees of each individual using the fusion operator @, and Lines 15–16 add the individual and the value to the set of answers if the value is not zero. Once all individuals have been processed, Line 20 orders the result (in decreasing value) and Line 21 filters the list to the keep the k highest values, before returning the query result.

Algorithm 1 Flexible Query Processing

```
Input: A knowledge graph $\mathcal{K}$
Input: A flexible query $\langle\{C_i\},\{P_i\},\{D_i\},\{H_i\},\diamond,@,k\rangle$
Output: $QR=\{<x,\alpha_x>|x\in\mathcal{KG},\alpha_x\in[0,1]\}$
 1: $QR\leftarrow\emptyset$
 2: $\{x\}\leftarrow getInstances(\mathcal{K},\{C_i\},\diamond)$
 3: $\{F_i\}\leftarrow getFuzzyDatatypes(\mathcal{K},\{D_i\})$
 4: for all $x\in\{x\}$ do
 5:     $r_i^x\leftarrow\emptyset$
 6:     for all $P_i\in\{P_i\}$ do
 7:         $y_i^x\leftarrow getPropertyValues(\mathcal{K},x,P_i)$
 8:         if $y_i^x\neq$ NULL then
 9:             $z_i^x\leftarrow z_i^x\cup\{getMembershipDegree(y_i^x,F_i)\}$
10:             $r_i^x\leftarrow applyModifier(z_i^x,H_i)$
11:         end if
12:     end for
13:     if $|r_i^x|=|P_i|$ then
14:         $\alpha_x\leftarrow fusion(r_i^x,@)$
15:         if $\alpha_x>0$ then
16:             $QR\leftarrow QR\cup\langle x,\alpha_x\rangle$
17:         end if
18:     end if
19: end for
20: $QR\leftarrow order(QR)$
21: $QR\leftarrow slice(QR,1,k)$
22: return $QR$
```

More precisely, we can solve the flexible query by performing the following steps:

1. To retrieve the candidates (members of the classes) and the values of the data properties when the retrieval operator is the conjunction, we retrieve the members of all classes by using the following SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?x ?Y_1 ... ?Y_n
WHERE {
   ?x rdf:type/rdfs:subClassOf* C_1 .
   ...
   ?x rdf:type/rdfs:subClassOf* C_N .
   ?x P_1 ?Y_1 .
   ...
   ?x P_n ?Y_n .
}
```

The result of this step is a list of tuples $\langle x, y_1^x, ..., y_n^x \rangle$. Note that this query not only retrieves the direct instances of the concepts $C_1, ..., C_n$, but also their indirect instances (i.e., including the instances of some of their subclasses). It could also be extended to retrieve values $\texttt{Y}_i$ linked to `?x` via a subproperty of P_i, or to infer the classes of `?x` via some domain or role restrictions.
Instead, if the retrieval operator is the union, we retrieve the members of some of the classes by using the following SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?x ?Y₁ ... ?Yₙ
WHERE {
   { { ?x rdf:type/rdfs:subClassOf* C₁ } UNION
   ...
     { ?x rdf:type/rdfs:subClassOf* C_N } } .
   ?x P₁ ?Y₁ .
   ...
   ?x Pₙ ?Yₙ .
}
```

Example 6. The next query retrieves instances of any of the classes `dbo:Motorcycle` and `dbo:Automobile`, together with the values of the properties `dbp:maxSpeedKmh`, `dbp:cruiseSpeedKmh`, and `dbp:emptyWeightKg`:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www .w3.org/2000/01/rdf-schema#>
PREFIX dbo:<http://dbpedia.org/ontology/>
PREFIX dbp:<http://dbpedia.org/property/>
SELECT DISTINCT ?x ?Y1 ?Y2 ?Y3
WHERE {
   { { ?x rdf:type/rdfs:subClassOf* dbo:Motorcycle } UNION
     { ?x rdf:type/rdfs:subClassOf* dbo:Automobile } } .
   ?x dbp:maxSpeedKmh ?Y1 ;
      dbp:cruiseSpeedKmh ?Y2 ;
      dbp:emptyWeightKg ?Y3
}
```

□

2. To retrieve the definition F_i of the input fuzzy datatype D_i for each $i \in \{1, ..., n\}$, we can use the following SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?Fᵢ
WHERE {
   Dᵢ rdf:type rdfs:Datatype .
   Dᵢ fuzzyLabel Fᵢ .
}
```

3. To compute, for each individual `?x` and each data property P_i, the membership degree of the value Y_i to the fuzzy datatype F_i, we can proceed as follows:

$$z_i^x = F_i(y_i^x),\ \forall i \in \{1,\dots,n\} \tag{1}$$

4. To apply the fuzzy hedge H_i to modify the membership degree for each `?x`, we compute the following expression:

$$r_i^x = H_i(z_i^x) \tag{2}$$

5. To aggregate all the values r_i^x corresponding to an individual `?x` into a single result α_x using the input fusion operator, we can compute the following values:

$$\alpha_x = @_{i\in\{1,\dots,n\}}(r_i^x) \tag{3}$$

6. We form a list of values $\langle ?x, \alpha_x \rangle$ sorted in decreasing order according to the value α_x and filtered to ensure that $\alpha_x > 0$ and to restrict to the top-k results (i.e., the individuals with the k highest values of α_x).

The process can be illustrated with the flowchart in Figure 2.

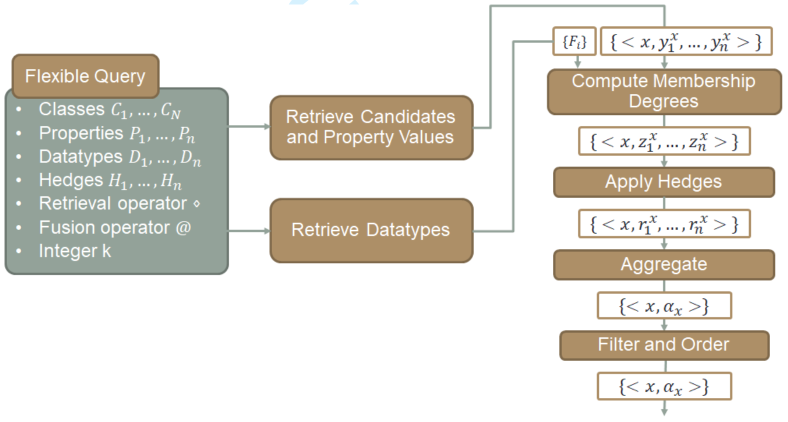

Flexible Query
Classes $C_1, \dots, C_N$
Properties $P_1, \dots, P_n$
Datatypes $D_1, \dots, D_n$
Hedges $H_1, \dots, H_n$
Retrieval operator ⋄
Fusion operator @
Integer k
Retrieve Candidates and Property Values
Retrieve Datatypes
$\{F_i\}$
$\{<x, y_1^x, \dots, y_n^x>\}$
Compute Membership Degrees
$\{<x, z_1^x, \dots, z_n^x>\}$
Apply Hedges
$\{<x, r_1^x, \dots, r_n^x>\}$
Aggregate
$\{<x, \alpha_x>\}$
Filter and Order
$\{<x, \alpha_x>\}$

Fig. 2: Flowchart of the query answering process algorithm

Example 7. Let us solve the query in Example 4 given the following set of triples on the domain of means of transportation:

```
@PREFIX dbo: <http://dbpedia.org/ontology/> .
@PREFIX dbr: < <http://dbpedia.org/resource/> .
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> .
```

```
dbo:Automobile rdfs:subclassOf dbo:MeanOfTransportation .
dbo:Motorcycle rdfs:subclassOf dbo:MeanOfTransportation .
dbr:Fiat_G.91Y rdf:type dbo:Automobile ;
               dbp:maxSpeedKmh "1110"^^xsd:decimal ;
               dbp:emptyWeightKg "3900"^^xsd:decimal ;
               dbp:cruiseSpeedKmh "630"^^xsd:decimal .
dbr:McDonnell_Douglas_A-4G_Skyhawk rdf:type dbo:Automobile ;
               dbp:maxSpeedKmh "1086"^^xsd:decimal ;
               dbp:emptyWeightKg "4581"^^xsd:decimal ;
               dbp:cruiseSpeedKmh "788"^^xsd:decimal .
dbr:Tupolev_Tu-95 rdf:type dbo:Automobile ;
                  dbp:maxSpeedKmh "925"^^xsd:decimal ;
                  dbp:emptyWeightKg "90000"^^xsd:decimal ;
                  dbp:cruiseSpeedKmh "710"^^xsd:decimal .
```

The first SPARQL query retrieves both dbr:Fiat_G.91Y, dbr:McDonnell_Douglas_A-4G_Skyhawk, and dbr:Tupolev_Tu-95, together with their values of maximum speed, cruise speeds, and empty weight. After that, the algorithm computes z_1,z_2 and z_3, and then $\alpha_x = (z_1 \cdot z_2 \cdot z_3)^2$. More precisely, the obtained values are:

?x	?Y1	?Y2	?Y3	z1	r1	z2/r2	z3/r3	α_x
dbr:Fiat_G.91Y	1110	3900	630	0.78	0.61	0.55	0.52	0.17
dbr:McDonnell_Douglas_A-4G_Skyhawk	1086	4581	788	0.83	0.69	0.21	0.85	0.12
dbr:Tupolev_Tu-95	925	90000	710	0.85	0.72	0.00	0.84	0.00

Finally, the result is the following ordered list of pairs:

⟨dbr:Fiat_G.91Y, 0.17⟩
⟨dbr:McDonnell_Douglas_A-4G_Skyhawk, 0.12⟩

□

The main advantage of this algorithm is that it is possible to reuse standard RDF language and SPARQL query endpoints, similarly as the authors in Huitzil et al. (2020, 2021) do for fuzzy ontologies.

Note that some fusion operators and fuzzy hedges can be expressed using SPARQL built-in functions (or inner expressions), allowing to perform more computations without relying on external processing. However, the resulting SPARQL query could be quite complex so as to be included here, and there are operators that cannot be expressed in standard SPARQL, such as the geometric mean (involving n-th roots).

4 Implementation

This section describes *FUKG* (FUzzy Knowledge Graphs)[5], a prototype tool implementing the algorithm discussed in the previous section.

[5] http://webdiis.unizar.es/~fbobillo/fukg

4.1 Main features

We have implemented *FUKG*, a prototype tool implementing the algorithm discussed in the previous section. FUKG has been implemented in Java and has a graphical user interface developed using Swing so that the user can easily provide all the relevant information.

The objective of the graphical user interface is to provide the relevant information in a transparent to the user way:

– Providing the URL of the endpoint storing the Knowledge Graph.
– Providing the names of the concepts, data properties, and fuzzy datatypes.
– Selecting the retrieval, fusion and hedge operators from a list of options built into the implementation.

To simplify the user interaction, FUKG GUI has implemented several optimizations of the process:

– The tool hides to the user the prefixes of the URIs and only the fragments are shown.
– Rather than asking the user to write the full URI of a concept/property or to select a concept/property from the Knowledge Graph (which requires showing all possible options, overwhelming the user in large Knowledge Graphs), FUKG uses an auto-complete mechanism. The user starts writing the fragment of the URI and the available options are shown.
– FUKG assumes that all fuzzy datatypes have a name obtained by concatenating a label and the corresponding property name. Possible values for the label are VeryHigh, High, Neutral, Low, and VeryLow. For example, for the property maxSpeedKmh, possible fuzzy datatypes are VeryHighmaxSpeedKmh, HighmaxSpeedKmh, NeutralmaxSpeedKmh, LowmaxSpeedKmh, and VeryLowmaxSpeedKmh.
– To use weighted average as the fusion operator, rather than asking the user to choose the weight associated to each property, FUKG makes it possible to choose the preferred properties, and they will have double importance (a double weight) than regular properties. If there are n properties and m of them are marked as favorite properties, the weight of each favorite property is $\frac{2}{n+m}$, and the weight of each non-favorite property is $\frac{1}{n+m}$.
– To use OWA as the fusion operator, rather than asking the user to choose as many weights as the number of properties, FUKG makes it possible to select a function, a fuzzy quantifier, to learn the weights using quantifier-guided aggregation.

FUKG has the following dependencies:

– Apache Jena and Jena Java API Carroll et al. (2004) [6] are used to submit SPARQL queries to a remote SPARQL endpoint.
– Fuzzy OWL 2 API is used to parse the definitions of the fuzzy datatypes to retrieve the function type and its parameters Bobillo and Straccia (2011).

[6] http://jena.apache.org

– AutoCompleter library[7] assists users to write the names of the classes and properties.

In some cases, we are interested at querying knowledge graphs which do not include fuzzy datatype definitions and it is not possible to update the knowledge graph which such definitions (e.g., we do not have write permission). To support querying such knowledge graphs, FUKG makes it possible to include the fuzzy datatype definitions in a local Fuzzy OWL 2 ontology file `datatypes.owl`, as illustrated in Example 8.

Example 8. A file `datatypes.owl` defining fuzzy datatype HighmaxSpeedKmh could have the following content:

```
Prefix(:=<http://Ontology.es/datatypes.owl/>)
Ontology(<http://Ontology.es/datatypes.owl>
  Declaration(AnnotationProperty(:fuzzyLabel))
  Declaration(Datatype(:HighmaxSpeedKmh))
  AnnotationAssertion(:fuzzyLabel :HighmaxSpeedKmh
    "<fuzzyOwl2 fuzzyType=\"datatype\">
    <Datatype type=\"triangular\" a=\"500\" b=\"1000\" c=\"1500\" />
    </fuzzyOwl2>"
  )
)
```

□

This file is also used to include the definitions of the RIM fuzzy quantifiers used to build OWA operators. We assume that their names are of the form *RIM1*, *RIM2*, etc.

FUKG supports the following operators:

– Retrieval operators: intersection and union.
– Fusion operators: minimum, product, maximum, weighted mean, and OWA. That is, it supports two t-norms, one t-conorm, and two aggregation operators.
– Fuzzy hedges: very, few (defined in Section 2.2), and none (so no fuzzy hedge is applied).

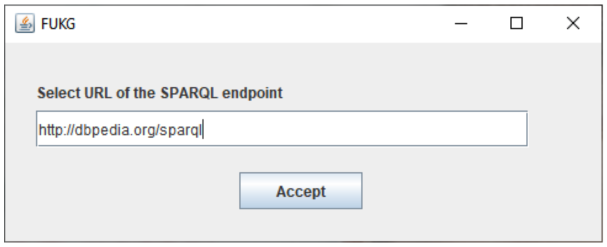
4.2 Description of the system

Selecting the knowledge graph. The first step is selecting the knowledge graph. To this end, a graphical interface asks the user to write the URL of the SPARQL endpoint of the knowledge graph. By default, the tool shows the URL of DBpedia SPARQL endpoint (`http://dbpedia.org/sparql`). This is illustrated in Figure 3. Then, FUKG submits an ASK query to the endpoint to check that it works:

```
ASK {?a ?b ?c}
```

If the answer to the ASK query is false, an error message is shown. Otherwise, the execution of the algorithm continues.

[7] http://serprogramador.es/autocompletar-en-java-swing/



Select URL of the SPARQL endpoint

http://dbpedia.org/sparql

Fig. 3: Selection of the knowledge graph

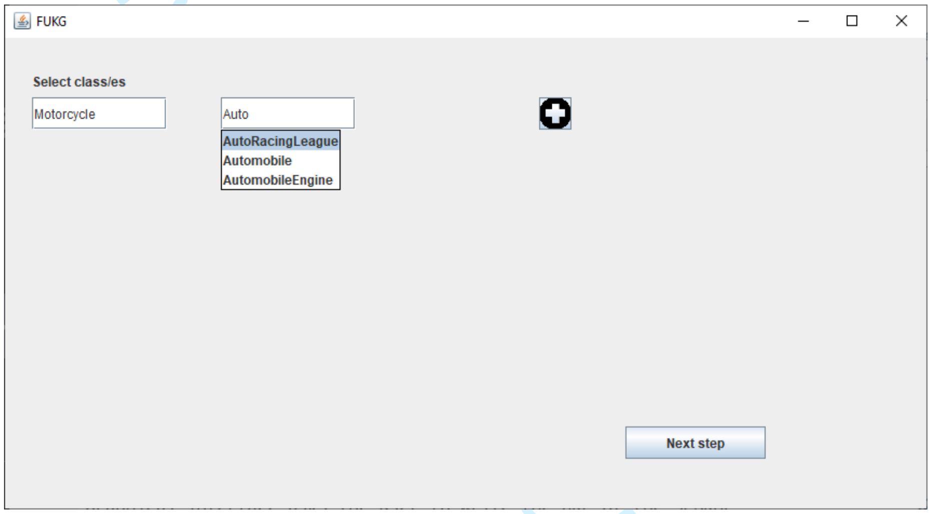


Fig. 4: Selection of a class using auto-complete

Selecting the classes. The next step is to retrieve the candidate classes that can be used in the query. To do so, FUKG retrieves all classes in the knowledge graph, which will be used later by the auto-complete utility. This is accomplished by using the following query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <https://www.w3.org/2002/07/owl#>
SELECT DISTINCT ?class
WHERE {
  { ?class rdf:type/rdfs:subClassOf* owl:Class }
  UNION
  {?x rdf:type/rdfs:subClassOf* ?class}
}
```

That is, FUKG retrieves both instances of owl:Class (or any of its subclasses), and objects of a rdf:type triple (or any of its subclasses). For example, in DBpedia, the query returns 1700 classes.

Once the candidate classes have been retrieved, the user can choose those of them that will be used in the flexible query using auto-complete and hiding the prefix of the URI, as illustrated in Figure 4. After selecting each class, the user can use a button to add a new class.

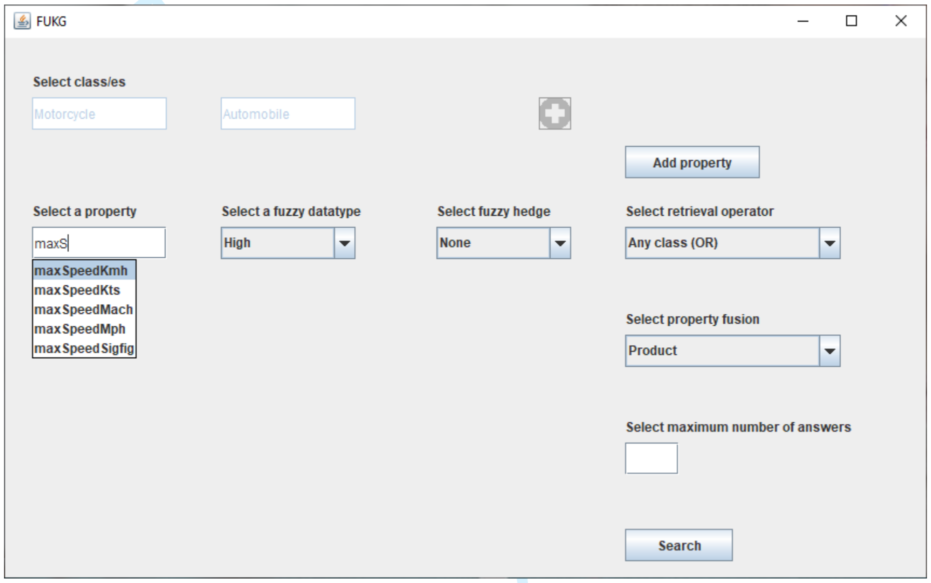


Fig. 5: Selection of a property using auto-complete

Selecting the properties. The next step is to retrieve the candidate properties that can be used in the query. The general idea is to retrieve all data properties for which any instance of the selected classes has a numerical value. If the user has selected classes $C1$,$C2$,...,CN, we use the following SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?property MIN(?valor)
WHERE {
 { { ?instance rdf:type/rdfs:subClassOf* C1 } UNION
   { ?instance rdf:type/rdfs:subClassOf* C2 } UNION
  ...
   { ?instance rdf:type/rdfs:subClassOf* CN } } .
 { ?instance ?property ?valor } FILTER isNumeric(?valor)
}
GROUP BY ?property
```

After retrieving the results, FUKG checks for errors. Typically, a knowledge graph may have some erroneous values for some individuals. For example, in DBpedia, property `dbp:aspectRatio` usually has numerical values, but sometimes it has an empty value *":: (en)"*. FUKG also checks that the values are indeed numeric (using `parseDouble` method from the `Double` class) and non-numerical values are discarded. In fact, using the SPARQL endpoint, we noticed that boolean values (e.g., `"true"^^http://www.w3.org/2001/XMLSchema#boolean>` for property `dbp:isHandicappedAccessible` in DBpedia) are retrieved despite having used `isNumeric` SPARQL function.

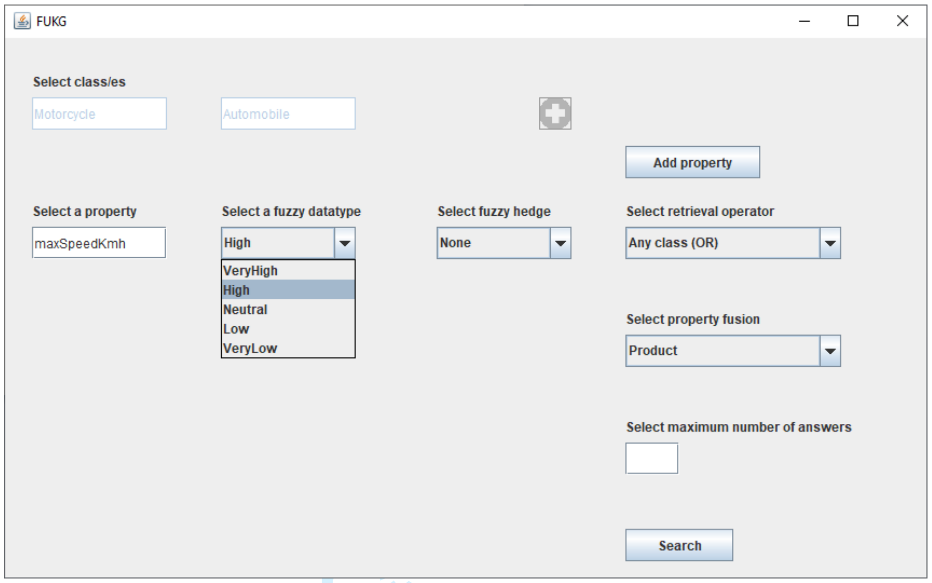
For example, to retrieve the properties when the user has selected DBpedia classes dbo:Motorcycle and dbo:Automobile, we use the following query, which returns 1275 properties after removing non-numerical variables:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?property MIN(?valor)
WHERE {
 { { ?instance rdf:type/rdfs:subClassOf* dbo:Motorcycle } UNION
   { ?instance rdf:type/rdfs:subClassOf* dbo:Automobile } } .
 { ?instance ?property ?valor } FILTER isNumeric(?valor)
}
GROUP BY ?property
```

Similarly as with the classes, when candidate properties have been retrieved, the user can choose those of them that will be used in the flexible query using auto-complete and hiding the prefix of the URI, as illustrated in Figure 5. In the case of elements with a different namespace but the same fragment, the namespace should also be shown to disambiguate. By clicking on a button, one can add a new property.

Selecting the fuzzy datatypes. For each of the selected properties, the user must choose a fuzzy datatype. To simplify the interface, for each property the user only has to choose one of the five possible labels (VeryHigh, High, Neutral, Low, and VeryLow), as illustrated in Figure 6. Then, the fuzzy datatype name is obtained by concatenating the label and the property name, and FUKG retrieves the fuzzy datatype definition from a local file if it exists. Note that it is not necessary that the local file includes the definitions of the five fuzzy datatypes for each property; but those fuzzy datatypes used in the flexible queries must be defined in the file.

Selecting the fuzzy hedges. For each of the selected properties, the user must choose a fuzzy hedge. Currently, the user can choose one of the following three possible labels (None, Very, and Few), as illustrated in Figure 7. By default, no fuzzy hedge is applied.



FUKG

Select class/es: Motorcycle, Automobile

Select a property: maxSpeedKmh

Select a fuzzy datatype: High

VeryHigh
High
Neutral
Low
VeryLow

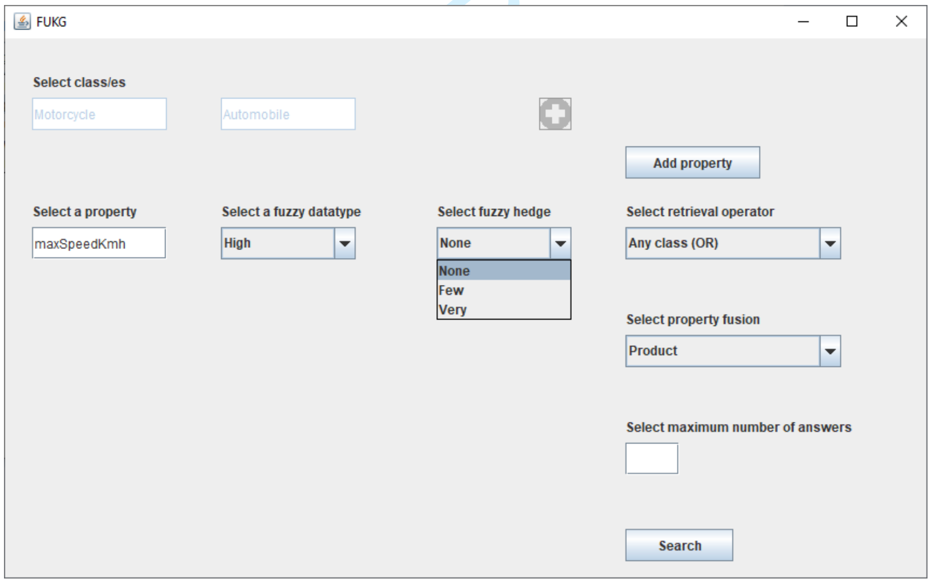
Select fuzzy hedge: None

Select retrieval operator: Any class (OR)

Select property fusion: Product

Select maximum number of answers:

Fig. 6: Selection of a fuzzy datatype



FUKG

Select class/es: Motorcycle, Automobile

Select a property: maxSpeedKmh

Select a fuzzy datatype: High

Select fuzzy hedge: None

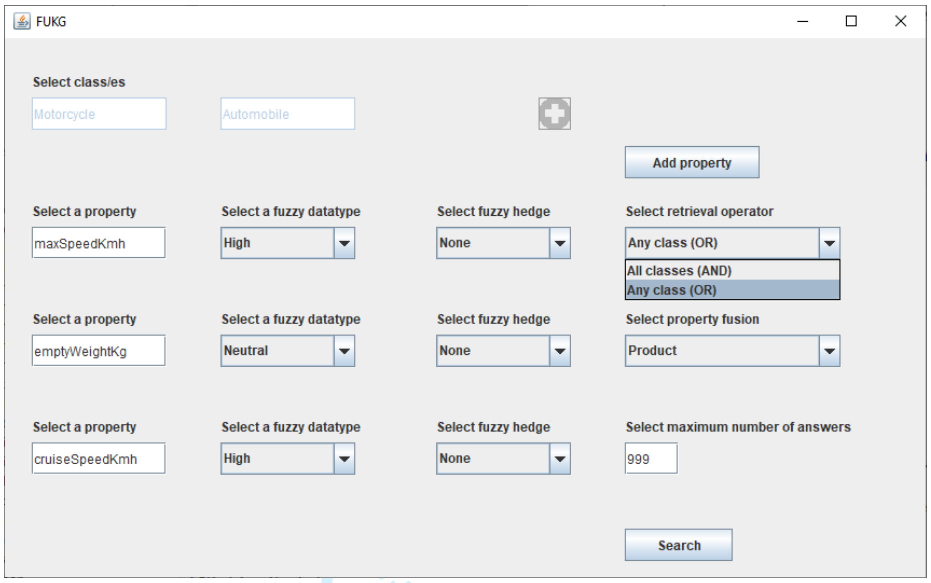
None
Few
Very

Select retrieval operator: Any class (OR)

Select property fusion: Product

Select maximum number of answers:

Fig. 7: Selection of a fuzzy hedge

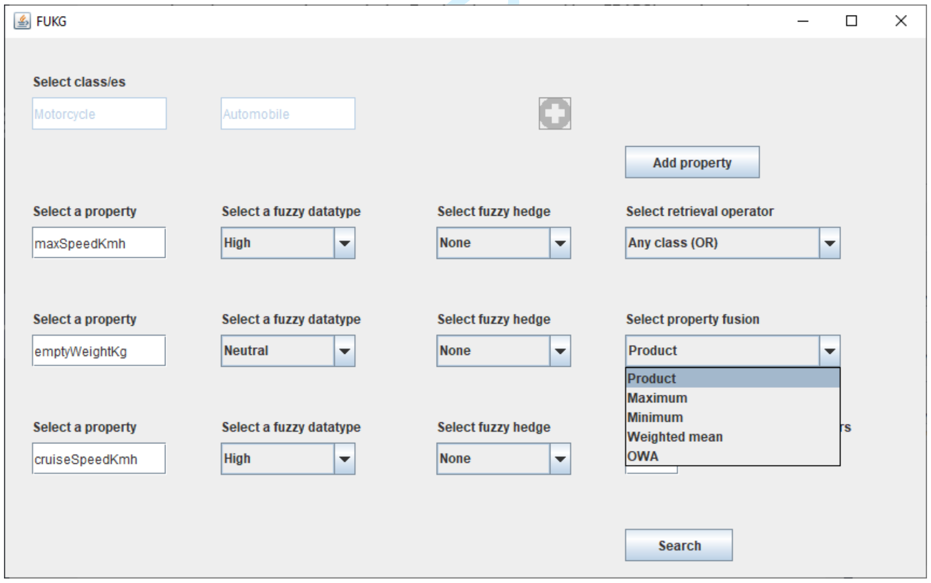


Select class/es: Motorcycle, Automobile

Select a property	Select a fuzzy datatype	Select fuzzy hedge	
maxSpeedKmh	High	None	Select retrieval operator: Any class (OR)
emptyWeightKg	Neutral	None	Select property fusion: Product
cruiseSpeedKmh	High	None	Select maximum number of answers: 999

Select retrieval operator options: All classes (AND); Any class (OR)

Fig. 8: Selection of a retrieval operator



Select class/es: Motorcycle, Automobile

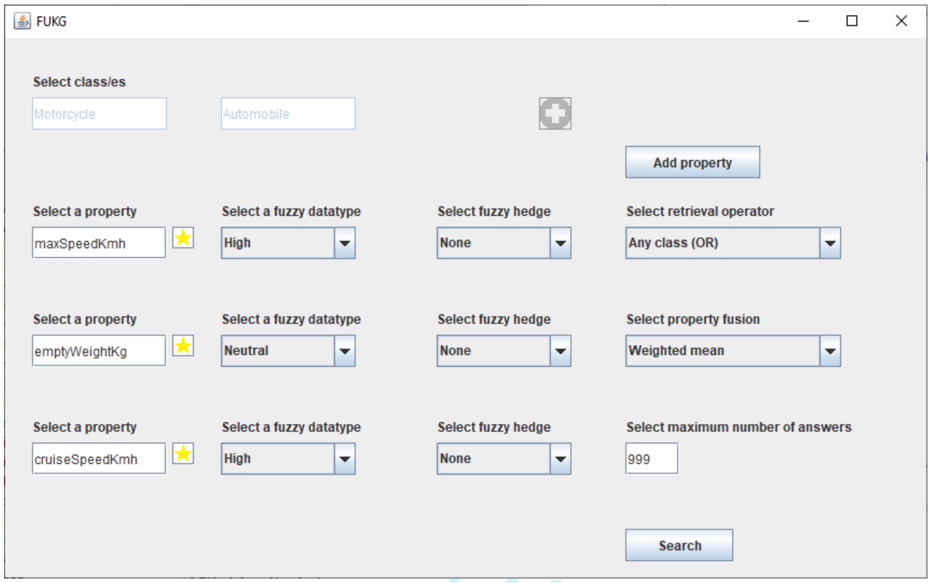
Select a property	Select a fuzzy datatype	Select fuzzy hedge	
maxSpeedKmh	High	None	Select retrieval operator: Any class (OR)
emptyWeightKg	Neutral	None	Select property fusion: Product
cruiseSpeedKmh	High	None	

Select property fusion options: Product; Maximum; Minimum; Weighted mean; OWA

Fig. 9: Selection of a fusion operator

Selecting the other parameters. The user also must specify the other parameters of the flexible query.

- Retrieval operator for the classes ($\diamond$): there is a list with two options: all classes or any class (Figure 8).



Select class/es: Motorcycle, Automobile

Select a property	Select a fuzzy datatype	Select fuzzy hedge	
maxSpeedKmh	High	None	Select retrieval operator: Any class (OR)
emptyWeightKg	Neutral	None	Select property fusion: Weighted mean
cruiseSpeedKmh	High	None	Select maximum number of answers: 999

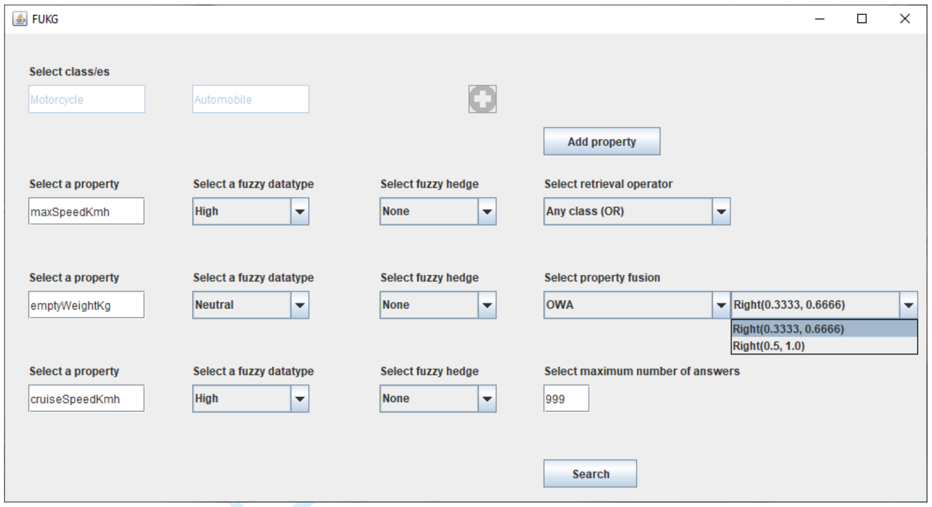
Fig. 10: Selection of preferred properties for the weighted mean

- Fusion operator (@): there is a list with five options: product, minimum, maximum, weighted mean, and OWA (Figure 9). The two latter options also require an additional action to define the weights: selecting preferred properties (for weighted mean, as shown in Figure 10) or selecting a fuzzy quantifier (for OWA, as shown in Figure 11, where *RIM1* is defined as **right**($^1/_3$,$^2/_3$), and *RIM2* is defined as **right**(0.5,1)).
- Maximum number of answers (k): a numerical value can be specified in a `JFormattedTextField`.

Submitting the query. Finally, the user can press a button (*"Search"*) and a new window with the results will be shown, as we will illustrate in the next section with a use case.

5 Use case

In this section, we will discuss a practical use case to illustrate the usefulness and flexibility of FUKG. In particular, we will use DBpedia as the source knowledge



Select class/es: Motorcycle, Automobile

Select a property: maxSpeedKmh — Select a fuzzy datatype: High — Select fuzzy hedge: None — Select retrieval operator: Any class (OR)

Select a property: emptyWeightKg — Select a fuzzy datatype: Neutral — Select fuzzy hedge: None — Select property fusion: OWA — Right(0.3333, 0.6666) [Right(0.3333, 0.6666) / Right(0.5, 1.0)]

Select a property: cruiseSpeedKmh — Select a fuzzy datatype: High — Select fuzzy hedge: None — Select maximum number of answers: 999

Fig. 11: Selection of a fuzzy quantifier to build the OWA weights

graph to retrieve information about means of transportation which are fast (in terms of maximum and cruising speed) and neither heavy nor light (in terms of empty weight). More precisely, we will look for either motorcycles or automobiles[8] with high maximum and cruising speeds and neutral empty weight.

We will propose four queries Q1–Q4 using different fuzzy fusion operators for the properties in order to show how they can affect the results of the queries:

- Classes: dbo:Motorcycle and dbo:Automobile
- Data properties: dbp:maxSpeedKmh, dbp:emptyWeightKg and dbp:cruiseSpeedKmh
- Fuzzy datatypes: HighmaxSpeedKmh, defined as **triangular**(500, 1000, 1500), NeutralemptyWeightKg, defined as **triangular**(1000,3000,5000), and HighcruiseSpeedKmh, defined as **triangular**(500,750,1000)
- Fuzzy hedges: none
- Retrieval operator: union $\vee$ (any of the classes)
- Fusion operator
  Q1: product
  Q2: weighted mean with equally preferred properties
  Q3: OWA with a fuzzy quantifier **right**($^1/_3$,$^2/_3$)
  Q4: maximum
- Number of results: 999

These queries are similar to the query in Example 4, but we considered alternative fusion operators and discarded the fuzzy hedge. To provide an intuitive description of the four queries, while Q1 computes a product of the satisfaction degrees, Q2 offers a

[8] Note that the results include some aircrafts which are asserted to be instances of dbo:Automobile in DBpedia.

softened version of Q1, where "failures" (0 values) are not so penalized, Q3 can be seen as a filter/equalizer (discarding the higher and the lower value), and Q4 provides an optimistic view by focusing on the most salient value (i.e., the highest satisfaction degree).

The first query (Q1) uses the product fusion operator. It requires that all the satisfaction degrees of the flexible restriction associated to the properties are greater than zero; if any of the values is zero, the product becomes zero. Furthermore, the more properties there are in the query, the more likely it is for any individual that one of these properties will be 0. Therefore, only 2 individuals are retrieved, as shown in Figure 12a. The first individual is dbr:Fiat_G.91Y and the second one is dbr:McDonnell_Douglas_A-4G_Skyhawk.

In the second query (Q2), the product fusion operator has been replaced with the weighted mean. Since all properties are equally preferred, a weight of $\frac{1}{3}$ is associated to each property. Now, the number of answer increases to 21, as a single degree zero in one of the properties does not imply that the global result is zero. Furthermore, values are higher in general, as a product of three numbers in [0,1] is replaced with a sum of three numbers in [0,1]. As we can see in Figure 12b, the first two individuals are the same ones as in the previous example, but now they have a different order: the first individual is dbr:McDonnell_Douglas_A-4G_Skyhawk and the second one is dbr:Fiat_G.91Y. For the other 19 individuals which do not appear in the results of the previous query, we can guess that the satisfaction degree of one of the constraints is zero.

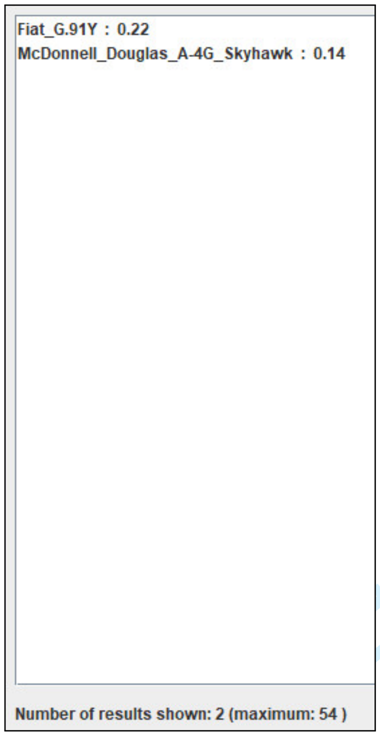
In the third query (Q3), the fusion operator is OWA. Because there are three values to be aggregated, according to Example 2, the selected fuzzy quantifier returns the vector of weights [0,1,0], giving all the importance to the value to be aggregated with the intermediate value (discarding the maximum and the minimum). In this case, the number of results is the same (21), but the order of the individuals and their degrees are different. Now, individuals with more balanced values are ranked higher. For example, dbr:Tupolev_Tu-95 is now ranked as the first individual, although it was the fifth result for Q2 and was not a result for Q1. The results are shown in Figure 13a.

Finally, the last query (Q4) uses maximum as a fusion operator. The number of results is the same one, but degrees are much higher. In general, the more properties in the query, the more likely it is for any individual to have a satisfaction degree of 1.0 (or close to 1.0) for some of the properties. The ranking of individuals is very different to the previous examples, as it can be seen in Figure 13b. For example, the first individual, dbr:Fokker_F.VII, was the seventh one in queries Q2 and Q3 and was not a result for Q1.

To conclude, let us mention that FUKG solved the previous queries fast (about 0.2 seconds per query), although the initialization time was higher: retrieving the list of classes from DBpedia took about 30 s in the first query (the next queries used cache and takes about 1 second), and retrieving the list of properties took about 20 s.

6 Related work

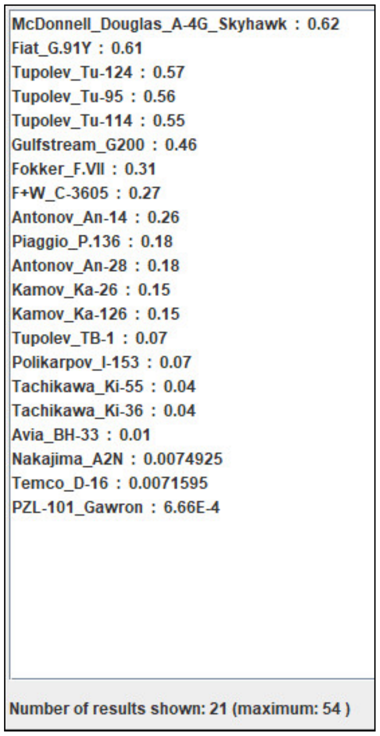
For a long time, extensions of Semantic Web technologies with different formalisms to represent and manage uncertainty have been proposed in the literature. In particular, the combination of Fuzzy Logic and Semantic Web has received a notable attention Sanchez (2006), Straccia (2013). Most of the work has focused on fuzzy on-



```
Fiat_G.91Y : 0.22
McDonnell_Douglas_A-4G_Skyhawk : 0.14

Number of results shown: 2 (maximum: 54 )
```

(a) Q1 (product)



```
McDonnell_Douglas_A-4G_Skyhawk : 0.62
Fiat_G.91Y : 0.61
Tupolev_Tu-124 : 0.57
Tupolev_Tu-95 : 0.56
Tupolev_Tu-114 : 0.55
Gulfstream_G200 : 0.46
Fokker_F.VII : 0.31
F+W_C-3605 : 0.27
Antonov_An-14 : 0.26
Piaggio_P.136 : 0.18
Antonov_An-28 : 0.18
Kamov_Ka-26 : 0.15
Kamov_Ka-126 : 0.15
Tupolev_TB-1 : 0.07
Polikarpov_I-153 : 0.07
Tachikawa_Ki-55 : 0.04
Tachikawa_Ki-36 : 0.04
Avia_BH-33 : 0.01
Nakajima_A2N : 0.0074925
Temco_D-16 : 0.0071595
PZL-101_Gawron : 6.66E-4

Number of results shown: 21 (maximum: 54 )
```

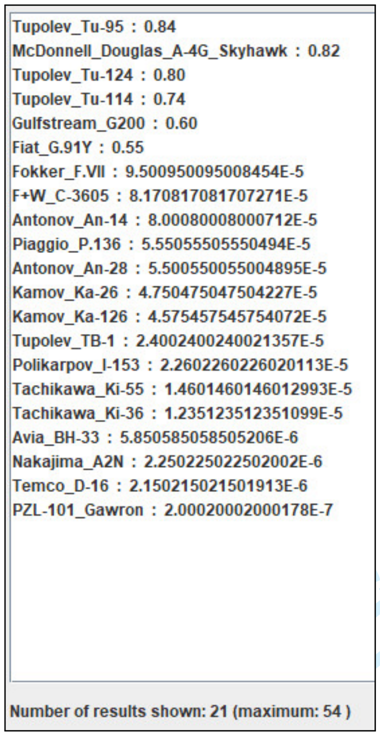
(b) Q2 (weighted mean)

Fig. 12: Results for Q1 (product) and Q2 (weighted mean)

tologies Lukasiewicz and Straccia (2008), Zhang et al. (2016) or fuzzy Description Logics Bobillo et al. (2015), as the main formalism behind fuzzy ontologies, including fuzzy extensions of fuzzy ontology languages such as OWL 2 Bobillo and Straccia (2011).

The literature also includes previous efforts studying the combination of Fuzzy Logic and RDF language. However, existing approaches focus on fuzzy axioms, whereas our approach focus on fuzzy datatypes. Indeed, none of the previous approaches addresses the representation of fuzzy datatypes:

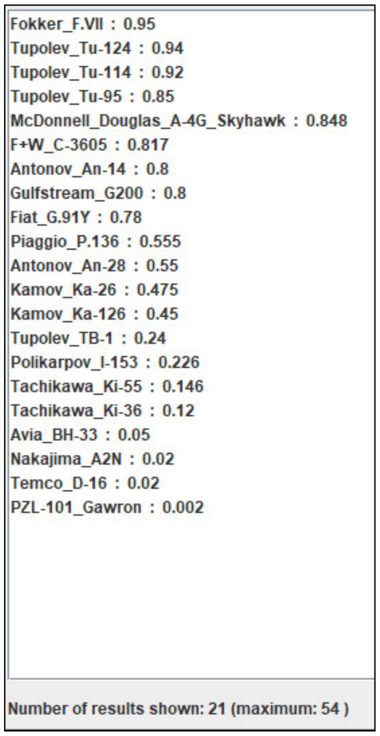
– Vaneková et al. proposed, to the best of our knowledge, the oldest fuzzy extension of RDF Vaneková et al. (2005). Their approach makes it possible to represent a fuzzy fact of the form "a resource s belongs to a fuzzy set f with degree α", with $\alpha \in (0,1]$, using an RDF triple of the form $\langle s,f,\alpha\rangle$. For example, ⟨aircraft001,FastAircraft,0.9⟩ denotes that aircraft001 belongs to the class of fast aircrafts with a high degree (0.9), so it is a fast aircraft.
  While this approach makes it possible to use a single triple reusing standard RDF, there is an implicit relationship type which is not being represented. For example, the authors do not represent the fact that aircraft001 is of type (rdf:type) aircraft, or the fact that aircraft001's speed is fast.



```
Tupolev_Tu-95 : 0.84
McDonnell_Douglas_A-4G_Skyhawk : 0.82
Tupolev_Tu-124 : 0.80
Tupolev_Tu-114 : 0.74
Gulfstream_G200 : 0.60
Fiat_G.91Y : 0.55
Fokker_F.VII : 9.500950095008454E-5
F+W_C-3605 : 8.170817081707271E-5
Antonov_An-14 : 8.00080008000712E-5
Piaggio_P.136 : 5.5505550555050494E-5
Antonov_An-28 : 5.500550055004895E-5
Kamov_Ka-26 : 4.750475047504227E-5
Kamov_Ka-126 : 4.575457545754072E-5
Tupolev_TB-1 : 2.4002400240021357E-5
Polikarpov_I-153 : 2.2602260226020113E-5
Tachikawa_Ki-55 : 1.4601460146012993E-5
Tachikawa_Ki-36 : 1.235123512351099E-5
Avia_BH-33 : 5.850585058505206E-6
Nakajima_A2N : 2.250225022502002E-6
Temco_D-16 : 2.150215021501913E-6
PZL-101_Gawron : 2.00020002000178E-7

Number of results shown: 21 (maximum: 54 )
```

(a) Q3 (OWA)



```
Fokker_F.VII : 0.95
Tupolev_Tu-124 : 0.94
Tupolev_Tu-114 : 0.92
Tupolev_Tu-95 : 0.85
McDonnell_Douglas_A-4G_Skyhawk : 0.848
F+W_C-3605 : 0.817
Antonov_An-14 : 0.8
Gulfstream_G200 : 0.8
Fiat_G.91Y : 0.78
Piaggio_P.136 : 0.555
Antonov_An-28 : 0.55
Kamov_Ka-26 : 0.475
Kamov_Ka-126 : 0.45
Tupolev_TB-1 : 0.24
Polikarpov_I-153 : 0.226
Tachikawa_Ki-55 : 0.146
Tachikawa_Ki-36 : 0.12
Avia_BH-33 : 0.05
Nakajima_A2N : 0.02
Temco_D-16 : 0.02
PZL-101_Gawron : 0.002

Number of results shown: 21 (maximum: 54 )
```

(b) Q4 (maximum)

Fig. 13: Results for Q3 (OWA) and Q4 (maximum)

– Similarly, M. Mazzieri and A. F. Dragoni proposed to replace RDF triples with quadruples of the form $\langle s,p,o,\alpha\rangle$ Mazzieri and Dragoni (2008), where $\alpha \in (0,1]$ quantifies the partial fulfillment of the RDF triple $\langle s,p,o\rangle$. Thus, this approach focuses on the representation of relationships which partially hold. Depending on the property p, we can have concept assertions (rdf:type), subclass axioms (rdfs:subclassOf), subproperty axioms (rdfs:subPropertyOf) or, more generally, object/data property assertions.
– A. E. A. Djebri discussed different approaches to annotate such statements of the form $\langle s,p,o,\alpha\rangle$: reification, n-ary properties, single named graph, singleton properties, and RDF-star Djebri (2022).
– Y. Lv et al. proposed a more general fuzzy extensions, with statements of the form $\langle \alpha_1/s,\alpha_2/p,\alpha_3/o,\alpha\rangle$, where the different degrees $\alpha_1,\alpha_2,\alpha_3$ reflect the membership degree of the subject, predicate, and object, respectively, to the RDF triple $\langle s,p,o,\alpha\rangle$ Lv et al. (2008). For example, this makes it possible to represent ⟨film001,0.8/starring,actor001⟩ or ⟨film001,genre,0.7comedy⟩. In the former case, the relationship (starring) is partial, whereas in the latter case the membership of the object (the degree of being a comedy) is partial.

The literature also includes previous efforts to query fuzzy RDF languages. However, our approach is more general, promotes the reuse of existing RDF knowledge graphs and SPARQL endpoints, and is suitable to query classical knowledge graphs via flexible datatypes:

- J. Z. Pan et al. proposed Fuzzy SPARQL, an extension of SPARQL designed to query fuzzy ontologies (but not fuzzy Knowledge Graphs) with statements of the form $\langle s,p,o,\alpha \rangle$ Pan et al. (2008). Fuzzy SPARQL uses comments to encode fuzzy restrictions, making it backwards compatible. For each triple in the SPARQL query, it is possible to add a degree of truth (restricting the value to be greater or equal than it, or exactly equal than it). This approach does not consider fuzzy datatypes or fuzzy hedges. Furthermore, although it is possible to specify a fusion operator type to combine the satisfaction degrees of each part of the query, the description of fusion operators is too general. For example, a possible option is "AGGREGATION", but the concrete aggregation operator is not specified.
- U. Straccia studied a fuzzy extension of ρDF (a fragment of RDF Schema) with statements of the form $\langle s,p,o,\alpha \rangle$ but no fuzzy datatypes Straccia (2009). He proposed a novel reasoning algorithm and fuzzy conjunctive queries over a fuzzy graph (which can include fuzzy triples), and assignments involving fuzzy membership functions and fusion operators, e.g., $q(x,s) \leftarrow \langle x, rdf:type, Aircraft \rangle \wedge \langle x, speed, y \rangle \wedge \{s := s1 \cdot Fast(y)\}$ Straccia (2009). The approach also includes a novel reasoning algorithm. However, this approach does not detail how to represent the syntax of the fuzzy datatypes (we instead use Fuzzy OWL 2 datatypes already represented in the RDF graph) and does not consider fuzzy hedges. Our approach could be friendlier to the user as there is no need to write such complex queries.
- J. Cheng et al. proposed f-SPARQL as another fuzzy extension of SPARQL where FILTER clauses can include fuzzy terms Cheng et al. (2010). For example, restrictions of the form $(?speed = fast)$ or $(?speed\ closeTo\ 200)$ are possible. Such restrictions can also include fuzzy hedges. The user can specify the relative importance of the RDF triples by assigning a different weight to each of triple. To answer the queries, fuzzy queries are translated into classical SPARQL queries by using α-cuts of the fuzzy sets, and the α values are the same user weights. Therefore, if the user does not specify a preference, $fast$ would be replaced by its core (i.e., the set of elements which belong to the fuzzy set with degree 1). Using α-cuts causes, for example, that $?speed = fast$ with a weight 0.8 would be replaced by a restriction of the form $?speed > 120$, where 120 being the minimal value such that $\mu_{fast}(120) = 0.8$. In all cases, fuzziness is removed before solving the SPARQL queries and thus without taking into account the real values in the knowledge base. Instead, our approach applies fuzziness to the real values retrieved by the SPARQL endpoint. Furthermore, there is no way to provide the definitions of the fuzzy membership functions (e.g., $fast$ or $closeTo$), and general fusion operators are not supported.
- O. Pivert et al. proposed yet another fuzzy extension of SPARQL called FURQL (Fuzzy RDF Query Language) Pivert et al. (2016), allowing fuzzy properties (that partially hold) and fuzzy datatypes in queries over a fuzzy RDF graph with statements of the form $\langle s,p,o,\alpha \rangle$. However, although fuzzy datatypes are used

in the examples, it is not discussed how to represent them in the graph, it is not possible to specify the fusion operator, and fuzzy hedges are not supported.

Last but not least, it is worth to mention a previous approach to answer classical queries over classical RDF using fuzzy logic operators Chen et al. (2022). X. Chen et al. proposed FuzzQE (Fuzzy Query Embedding), a fuzzy logic based embedding framework to answer classical queries over KGs. Supported queries are expressed using First Order Logic in disjunctive normal form. Then, they are embedded as fuzzy vectors, and Boolean operators are implemented using t-norms, t-conorms (maximum and probabilistic sum) and fuzzy negations (standard negation. In particular, FuzzQE supports two t-norms (minimum and product), two t-conorms (maximum and probabilistic sum), and one negation (standard). Compared to our approach, it does not support fuzzy datatypes (in query representation), fuzzy hedges or aggregation operators (in query answering). It is also worth to note that FuzzQE supported fuzzy operators are usually combined using Gödel negation, rather than with standard negation Straccia (2013).

7 Conclusions and future work

In this paper we have proposed a Fuzzy Logic based approach to answer flexible queries over Knowledge Graphs. The user not only specifies the classes of the individuals to be retrieved but also some flexible constraints on the values of the data properties, described using fuzzy sets.

The main advantage of our approach is that it makes it possible to reuse existing RDF graphs and SPARQL endpoints, building a fuzzy layer on top of them. An important novelty with respect to the previous work is that our approach supports Fuzzy OWL 2 datatypes to describe fuzzy membership functions. Interestingly, Fuzzy OWL 2 datatypes also makes it possible to reuse standard OWL 2 ontology language. Moreover, it is a very general approach that supports different choices of fuzzy operators: t-norms, t-conorms, aggregation operators, and fuzzy hedges.

Our flexible query answering algorithm has been implemented in the FUKG system, with an intuitive user graphical interface to avoid requiring users to deal with a concrete syntax and to assist users at several steps, for example, using auto-complete when typing the names the entities or using strategies to avoid writing numerical weights. We have also shown a use case to illustrate the capabilities of the system and the possibility to aggregate information in different ways by selecting different fusion operators. In our preliminary experiments, FUKG answers query over DBpedia in a few seconds.

Some limitations of FUKG are that it cannot support fuzzy axioms in the knowledge graph or fuzzy terms in the query, and that missing values do not receive any special attention. Therefore, a possible idea for the future work is to extend the tool to deal with missing values, rather than requiring that there is a value for each data property in the query. A possible solution is to aggregate only the values that are available. This was the approached followed in Huitzil et al. (2020), where the weights of the weighted mean and OWA are adapted for each individual to the number of criteria to be aggregated.

Future work could also include the application of FUKG to more real applications and to perform an empirical evaluation of the performance of the tool on such scenarios. Furthermore, the tool could be extended to support more fusion operators, fuzzy quantifiers, or fuzzy hedges. Indeed, thanks to the modular design of the system, adding new operators is very easy.

Acknowledgments

C. Bobed and F. Bobillo were supported by the I+D+i project PID2020-113903RB-I00, funded by MCIN/AEI/10.13039/501100011033, and by DGA/FEDER. I. Huitzil was partially supported by the Italian MUR under the PRIN project PINPOINT Prot. 2020FNEB27, CUP H45E21000210001.

Bibliography

Baader, F., Calvanese, D., McGuinness, D., Nardi, D. and Patel-Schneider, P. F.: 2003, *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press.

Bizer, C., Heath, T. and Berners-Lee, T.: 2009, Linked data - the story so far, *International Journal on Semantic Web and Information Systems* **5**(3), 1–22.

Bobillo, F.: 2016, The role of crisp elements in fuzzy ontologies: The case of fuzzy OWL 2 EL, *IEEE Transactions on Fuzzy Systems* **24**, 1193–1209.

Bobillo, F., Cerami, M., Esteva, F., García-Cerdaña, À., Peñaloza, R. and Straccia, U.: 2015, Fuzzy description logics, *in* P. Cintula, C. Fermüller and C. Noguera (eds), *Handbook of Mathematical Fuzzy Logic Volume III*, Vol. 58 of *Studies in Logic, Mathematical Logic and Foundations*, College Publications, chapter XVI, pp. 1105–1181.

Bobillo, F., Delgado, M., Gómez-Romero, J. and Straccia, U.: 2012, Joining Gödel and Zadeh fuzzy logics in fuzzy description logics, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **20**(4), 475–508.

Bobillo, F. and Straccia, U.: 2011, Fuzzy ontology representation using OWL 2, *International Journal of Approximate Reasoning* **52**(7), 1073–1094.

Brickley, D. and Guha, R. V.: 2014, RDF Schema 1.1, *W3C recommendation*, W3C. http://www.w3.org/TR/rdf-schema/.

Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A. and Wilkinson, K.: 2004, Jena: Implementing the semantic web recommendations, *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters*, ACM, pp. 74—-83.

Chen, X., Hu, Z. and Sun, Y.: 2022, Fuzzy logic based logical query answering on knowledge graphs, *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022)*, AAAI Press, pp. 3939–3948.

Cheng, J., Ma, Z. M. and Yan, L.: 2010, f-SPARQL: A flexible extension of SPARQL, *Proceedings of the 21st International Conference on Database and Expert Systems Applications (DEXA 2010), Part I*, Vol. 6261 of *Lecture Notes in Computer Science*, Springer, pp. 487–494.

Cuenca-Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P. and Sattler, U.: 2008, OWL 2: The next step for OWL, *Journal of Web Semantics* **6**(4), 309–322.

Djebri, A. E. A.: 2022, *Uncertainty Management for Linked Data Reliability on the Semantic Web*, PhD thesis, Université Côte D'Azur. http://hal.archives-ouvertes.fr/tel-03679118.

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutiérrez, C., Kirrane, S., Labra Gayo, J. E., Navigli, R., Neumaier, S., Ngonga Ngomo, A.-C., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J. F., Staab, S. and Zimmermann, A.: 2021, *Knowledge Graphs*, number 22 in *Synthesis Lectures on Data, Semantics, and Knowledge*, Morgan & Claypool.

Huitzil, I., Alegre, F. and Bobillo, F.: 2020, GimmeHop: A recommender system for mobile devices using ontology reasoners and fuzzy logic, *Fuzzy Sets and Systems* **401**, 55–77.

Huitzil, I., Molina-Solana, M., Gómez-Romero, J. and Bobillo, F.: 2021, Minimalistic fuzzy ontology reasoning: An application to Building Information Modeling, *Applied Soft Computing* **103**, 107158.

Klir, G. J. and Yuan, B.: 1995, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice-Hall.

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., Van Kleef, P., Auer, S. et al.: 2015, Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia, *Semantic web* **6**(2), 167–195.

Lukasiewicz, T. and Straccia, U.: 2008, Managing uncertainty and vagueness in Description Logics for the Semantic Web, *Journal of Web Semantics* **6**(4), 291–308.

Lv, Y., Ma, Z. M. and Yan, L.: 2008, Fuzzy RDF: A data model to represent fuzzy metadata, *Proceedings of the 17th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2008*, IEEE, pp. 1439–1445.

Mazzieri, M. and Dragoni, A. F.: 2008, A fuzzy semantics for the resource description framework, *Uncertainty Reasoning for the Semantic Web I*, Vol. 5327 of *Lecture Notes in Computer Science*, Springer, pp. 244–261.

Pan, J. Z., Stamou, G., Stoilos, G., Thomas, E. and Taylor, S.: 2008, Scalable querying service over fuzzy ontologies, *Proceedings of the 17th International World Wide Web Conference (WWW 2008)*, pp. 575–584.

Pérez, J., Arenas, M. and Gutiérrez, C.: 2006, The semantics and complexity of SPARQL, *Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, Vol. 4273 of *Lecture Notes in Computer Science*, Springer, pp. 30–43.

Pivert, O., Slama, O. and Thion, V.: 2016, An extension of SPARQL with fuzzy navigational capabilities for querying fuzzy RDF data, *Proceedings of the 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2016)*, IEEE, pp. 2409–2416.

Sanchez, E. (ed.): 2006, *Fuzzy Logic and the Semantic Web*, Vol. 1 of *Capturing Intelligence*, Elsevier Science.

Schreiber, G. and Raimond, Y.: 2014, RDF 1.1 primer, *W3C working group note*, W3C. http://www.w3.org/TR/rdf11-primer.

Sirin, E., Parsia, B., Cuenca-Grau, B., Kalyanpur, A. and Katz, Y.: 2007, Pellet: A practical OWL-DL reasoner, *Journal of Web Semantics* **5**(2), 51–53.

Staab, S. and Studer, R.: 2004, *Handbook on Ontologies*, International Handbooks on Information Systems, Springer.

Straccia, U.: 2009, A minimal deductive system for general fuzzy RDF, *Proceedings of the 3rd International Conference on Web Reasoning and Rule Systems (RR 2009)*, Vol. 5837 of *Lecture Notes in Computer Science*, Springer, pp. 166–181.

Straccia, U.: 2013, *Foundations of Fuzzy Logic and Semantic Web Languages*, CRC Studies in Informatics Series, Chapman & Hall.

Vaneková, V., Bella, J., Gurský, P. and Horváth, T.: 2005, Fuzzy RDF in the semantic web: deduction and induction, *Proceedings of the 6th Workshop on Data Analysis (WDA 2005)*, pp. 16–29.

Yager, R. R.: 1988, On ordered weighted averaging aggregation operators in multicriteria decision making, *IEEE Transactions on Systems, Man and Cybernetics* **18**(1), 183–190.

Yager, R. R.: 1996, Quantifier guided aggregation using OWA operators, *International Journal of Intelligent Systems* **11**(1), 49–73.

Yagüe, J. F., Huitzil, I., Bobed, C. and Bobillo, F.: 2022, Flexible queries over knowledge graphs, *Proceedings of the 4th Iberoamerican and 3rd Indo-American Knowledge Graphs and Semantic Web Conference (KGSWC 2022)*, Vol. 1686 of *Communications in Computer and Information Science*, Springer, pp. 192–200.

Zadeh, L. A.: 1965, Fuzzy sets, *Information and Control* **8**, 338–353.

Zhang, F., Cheng, J. and Ma, Z.: 2016, A survey on fuzzy ontologies for the semantic web, *Knowledge Engineering Review* **31**(3), 278–321.