

# Allowing Users to Create Similarity Relations for Their Flexible Searches Over Databases

Mohammad Halim Deedar and Susana Muñoz-Hernández

Escuela Técnica Superior de Ingenieros Informáticos, Universidad Politécnica de Madrid, Campus de Montegancedo s/n, Boadilla del Monte, 28660  
`halim.deedar@alumnos.upm.es`, `susana@fi.upm.es`

**Abstract.** A bi-valued logic is not enough to make an intelligent search engine to give us the result for the queries like "I am looking for a cheap restaurant, Mediterranean food or similar type." With the integration of Fuzzy Logic and Logic Programming, we were able to model and pose flexible queries over databases. Therefore, we present a framework that allows users to pose their expressive queries based on defining similar relation criteria over various modern and conventional data formats such as JSON, SQL, CSV, XLS, and XLSX. The interest is in, for example, obtaining "drama movie" when asking for "romantic movie" (only if the similarity relation between drama and romantic movie is explicitly defined in the configuration file). The uses of similarity relation between values allow us to obtain more answers apart from the identical one. The searches that use two or more criteria are much more expressive and accurate. This framework provides the facility to define, modify and remove similarity relations from a user-friendly interface (without the need to be concern about the low-level syntax of the similarity criteria).

**Keywords:** Fuzzy logic · Similarity relation · Expressive searches · Search engine · framework.

## 1 Introduction

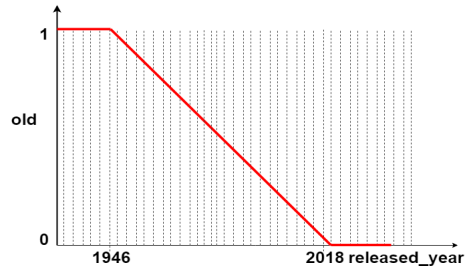
Assume a database storing information on films, containing entities such as film's name, genre, etc. A user wants to retrieve "a list of all the films existing in the film database which are very similar to a drama film." It is not functional to store fuzzy information with those fuzzy concepts (very similar, completely similar, cheap, far, etc.) in a database. However, it is not surprising that an end-user being a human to have that type of queries, as they always think and query in an expressive(fuzzy) way. Although searching in a fuzzy way not only gives us the exact information we are looking for, but it lets us to retrieve all the possible and available information close to the criteria which we have set for our query. For example, if we do a normal query (non-fuzzy) using any other standard query languages such as SQL to retrieve a movie from a film database (having a crisp information) which is of genre romance, we get a list of all the existing romance movies from the database but we may miss the ones who are

not explicitly defined as a romance movie but they are quite similar to romantic movie (because a drama movie sometime can be romantic too) and this could be the movie that the user is looking for. Therefore, we aim to provide a flexible database searching system that allows the user to pose their expressive queries over various data formats and get the best possible results.

A similarity is a relation between two real-world concepts. For a human being, it is not so difficult to decide how two things are similar or not to each other. In order to represent the real-world concepts as the human being understands, we need to introduce all the required information (or knowledge) explicitly. Therefore, for representing the real-world concept to the machines, we need a logic much more than a bi-valued logic (that defines if an individual belongs to a set or not by using only two value which is "true/false" or "yes/no").

Fuzzy logic [19, 20] has substantiated its capability devoted to the management of vague information in a different number of applications (such as control systems, database or expert systems). The integration of Fuzzy Logic and pure logic programming [9] provides Fuzzy Logic Programming with the capability of dealing with ambiguity and approximate reasoning. This integration provided us the facility to program the machines in order to understand the fuzzy characteristics (it is cold), fuzzy rules (if it is cold, turn on the heater) and fuzzy actions (since it is not too cold, turn on the heater at medium degree). Therefore, in our approach the representation of similarity relation between two real-world concepts is provided using the integration of Prolog which is more declarative and a successful programming language for representing knowledge, and the Fuzzy Logic that defines not only if an individual belongs to a set or not, but it also provides us the degree of its belonging from that set. Supposing, a film database and the definition for the criteria "old" in (Fig. 1), and the question "is film X is old?" with fuzzy logic we can deduce that Casablanca is "very" old, The Godfather is "almost" old, Thor is "hardly" old, and Halloween is "not" old. We highlight the words "very," "almost," "hardly," and "not" because the usual answers for the query are "1", "0.9", "0.1" and "0" for the individuals Casablanca, The Godfather, Thor, and Halloween.

| film_name     | genre   | release_year |
|---------------|---------|--------------|
| Casablanca    | Romance | 1946         |
| The Godfather | Drama   | 1972         |
| Thor          | Action  | 2013         |
| Halloween     | Horror  | 2018         |



**Fig. 1.** film database content and old fuzzification criterion.

Many tools have been developed and implemented based on the fuzzy set theory introduced by Zadeh (as cited in [1]) to represent the fuzzy knowledge as Prolog-Elf system by Ishizuka and Kanai[7], the FRIL Prolog system by Baldwin et al.[1], the F-Prolog language by Li and Liu[8], the FuzzyDL reasoner by Bobillo and Straccia[2], the Fuzzy Logic Programming Environment for Research (FLOPER) by Morcillo and Moreno[12], the Fuzzy Prolog system by Guadarrama et al.[6], RFuzzy by Muñoz-Hernández et al.[13], and the theoretical frameworks as Vojtáš[17]. But we have used RFuzzy with priorities[14, 15] in our approach to add a link between fuzzy and non-fuzzy concept because we need our approach to have the capability to decide the preferred results among the ones provided by different rules, it does not matter if the last rule provides a result with higher truth value.

We aim to present a framework intelligent and flexible enough that allows users to perform fuzzy (expressive) queries over databases (having crisp data) adding the similarity relation criteria. To our knowledge, the works similar to ours are[18, 5, 3, 4]. The main distinctive characteristics of our approach in comparison to other ones are (1) that we do not force our relations to be equivalence (reflexive, symmetric, or transitive) for our similarity criteria.(2) We are not trying to evaluate the closeness (or similarity) between two fuzzy propositions. Our work is different: The item which is similar to the one we are looking for is returned as a result after computing its similarity value. (3) There is no limitation on posing the expressive query only over a single data format because our framework allows the users to pose their expressive query over conventional and modern data formats such as JSON, SQL, CSV, XLS, and XLSX. (4) We allow the end-users to define, modify and remove similarity relations between two concepts without knowing the low-level syntax of the function, which can reduce the gap between the database end-users and the logic programming users.

This paper is structured as follows: A brief details about the general RFuzzy rule syntax, the syntax for defining databases, and the introduction to the Similarity syntax and semantics are given in (Section 2). In (Section 3) we present the implementation of the framework (web application) including the details about the system architecture and the steps for defining, querying, and modification of Similarity relations, and the results of the query after posing it over the database. Finally, in (Section 4) we give our conclusions and the current work.

## 2 Syntax and semantics of Similarity relation

Before discussing the syntax and semantics of similarity relations of our approach, we include a brief detail about the general RFuzzy rule syntax (Section 2.1) and the syntax for database definition (Section 2.2) because by some means they are related to each other.

### 2.1 General RFuzzy rule syntax

The general structure that is used to define RFuzzy rules in according to a multi-adjoint logic semantics is shown in Eq. 1[14]

$$P(argsP_j, V_j) \xleftarrow{(Pr_j, V_{cj}) \&_i} @_j (Q_1 (argsQ_{1j}, V_{1j}), \dots (Q_n (argsQ_{nj}, V_{nj}))). \quad (1)$$

Where  $P$  is the predicate,  $j$  is one of the definitions from a set of definitions  $j \in [1, N]$  (where  $N$  is number of rules that we have to define for predicate  $P$ , and  $j$  identifies one of these rules).  $argsP_j$  are the arguments of  $P$  in the rule  $j$ , in the same way  $argsQ_i$  are the arguments of  $Q_i$  where  $i \in [1, n]$  and  $n$  is the number of elements of the body of the clause.  $V_i$  is the truth value of the  $Q_i(argsQ_i, V_i)$ .  $@_j$  is the aggregation operator of the rule  $j$ .  $V_{cj}$  is the credibility to calculate the truth value and  $Pr_j$  is the priority of  $j$  rule with respect to other rules of  $P$  definition.

The multi-adjoint algebra as presented in [11, 10] is used to give semantics to our framework. The purpose of using this structure is that it provides credibility for the rules that we give in our program. We highlight this point, so the reader knows why our approach is based on this structure and not some other. Since comprehensive details about the semantics can be found in the papers cited.

## 2.2 Database definition syntax

The syntax which is responsible for outlining the contents of a database into concepts that we can use in our searches is shown in Eq. 2 Where  $P$  is the name of the database table,  $A$  is its arity,  $N$  is the name assigned to a column (field) of the database table where values are of type  $T$ ,  $i \in [1, A]$  identifies each field of the table. We give an example in Eq. 3, to elucidate, that the film table has four columns, the first one is for the name assigned to each film, the second is for the year in which the films have released, the third is for duration of the films in minutes, and the last one is for the genre of each film whose value belongs to an enumerate range (comedy, thriller, drama, romantic, adventures, etc.).

$$define\_database(P/A, [(N_i, T_i)]). \quad (2)$$

$$\begin{aligned} &define\_database (film/4, \\ &[(film\_name, string\_type), \\ &(release\_year, integer\_type), \\ &(duration\_in\_minutes, integer\_type), \\ &(genre, enum\_type)]). \end{aligned} \quad (3)$$

The web interface and setters/getters obtain plenty of information from the database definition.

## 2.3 Similarity definition between values

The syntactical construction which is used for modeling a similarity relation between the values of a field is shown in Eq. 4. Where  $P$  and  $N$  are the same as in Eq. 2,  $V1$  and  $V2$  are the possible values for the column  $N$  of table  $P$ , and

$TV$  is the truth value (a float number) that we assign to the similarity between  $V1$  and  $V2$ . To clarify, we give an example in Eq. 5, saying that "romance films are 0.7 similar to drama films".

$$similarity\_between(P, N, [(V1), N(V2), TV]). \quad (4)$$

$$similarity\_between(film, genre(romance), \quad (5) \\ genre(drama), 0.7).$$

The semantic for defining conditioned similarity is shown in Eq. 6 where  $P$ ,  $N$ ,  $TV$ ,  $V1$ , and  $V2$  are the same as in Eq. 4.  $p$  is the priority,  $v$  is the credibility,  $\&_i$  is the product, and  $COND$  is a bi-valued condition. To clarify, we give an example in Eq. 7, saying that the *films* with the genre *drama* are "very similar" to the films with the genre "romance" by "0.9" degree and with the credibility of "1" if the films are directed by *Richard*. This means, with the help of this structure we can define the similarity relation between two values based on a condition with the credibility of "1". For example, after posing a query for searching the similarity degree between "drama" and "romance" films, we may obtain multiple films of type "drama" and "romance", but with the help of the syntax in (Eq. 7) with a condition and the credibility value a user can deduce that the films which are directed by "Richard" (a film director) their similarity relation of being "drama" and "romantic" are more credible in comparison to the rest of the films.

$$similarity(P(N(V1, V2))) \xleftarrow{(p,v)\&_i} TV \text{ if } COND \quad (6)$$

$$similarity(film(genre(drama, romance))) \quad (7) \\ \xleftarrow{(1,1).prod} 0.9 \text{ if } director = "Richard".$$

## 2.4 Synonyms definition

The syntactical constructions for defining synonyms based on the similarity relations between fuzzy predicates is shown in Eq. 8, where  $fPredName$  is the name of the fuzzy predicate (expensive, cheap, etc.),  $P$  is the same as in Eq. 4,  $credOp$  is the operator (product by default), and  $credVal$  is the credibility (a type float number, which is 1 by default). To clarify, we give an example in Eq. 9, saying that "inexpensive is similar (or synonym) to cheap."

$$fPredName(P) : \sim \quad (8) \\ synonym\_of(fPredName2(P), credOp, credVal).$$

$$inexpensive(restaurant) : \sim \quad (9) \\ synonym\_of(cheap(restaurant), prod, 1).$$

Attending to this definition, any query asking for inexpensive restaurants will return the cheap ones.

## 2.5 Similarity relation definition between concepts

With the syntax in Eq. 10, we can define a fuzzy predicate from another fuzzy predicate by defining the similarity relation between both fuzzy concepts when a condition is satisfied. We can define "inexpensive" from "cheap", "gorgeous" from "beautiful" etc. This will lead to getting a huge vocabulary for fuzzy searching without defining or storing a definition for all new words which does not exist in our vocabulary. In the syntax  $fPredName$  is the same as in Eq. 8, an individual is an element of the database for which we want to obtain the fuzzy value, and COND is the same as in Eq. 6. To clarify, we give an example in Eq. 11, where we can get the answer for the query "I am looking for an inexpensive car" for this query we will get the list of all the cars (stored in the database) which have been defined as a cheap. COND equal to true means that the similarity relation is always satisfied.

$$fPredName(individual) \xleftarrow{(p,v)\&i} fPredName2(individual) \text{ if } COND. \quad (10)$$

$$inexpensive(individual) \xleftarrow{(0,1),prod} cheap(individual) \text{ if true}. \quad (11)$$

## 3 Implementation details

Our proposed system is a combination of a web interface and a framework, developed mainly in Java, JavaScript, HTML, and AJAX. It uses RFuzzy package which is a Prolog library developed for Ciao Prolog[16]. The web interface is used to pose flexible queries over multiple conventional and modern data formats "JSON, SQL, CSV, XLS, and XLSX". The framework part of our system is used for defining the links between the crisp information stored in a database and the fuzzy concepts that we can use in a query. As far as we know, our system is novel in the idea of allowing the regular users (without having knowledge about database and programming) to create a link between the crisp information and the fuzzy concepts, and we allow them to define the similarity relation between concepts without knowing the low-level syntax of the criteria. Moreover, we provide them the facility to perform their flexible query over multiple data formats such as "JSON, SQL, CSV, XLS, and XLSX" (they do not need to have their database in a particular format such as "Prolog" which is not so familiar to the most regular users).

### 3.1 System architecture

The system architecture, which is shown in Fig. 2, has five main parts: (i) Criteria Definition module: through which users can define, modify, personalize, and remove the similarity relations. (ii) Flexible Query Engine: is the main part, where users can pose their flexible queries devoted similarity relation. (iii) Flexible query process module that takes the flexible queries as an input from the

users and provides the results as output after several computations of similarity relations. (iv) An engine (ciao prolog) that compiles our query for allowing the system to understand it and compute the corresponding similarity relations for it. (v) A database (configuration file) where data, similarity relations, quantifiers, modifiers, and negation operator are stored.

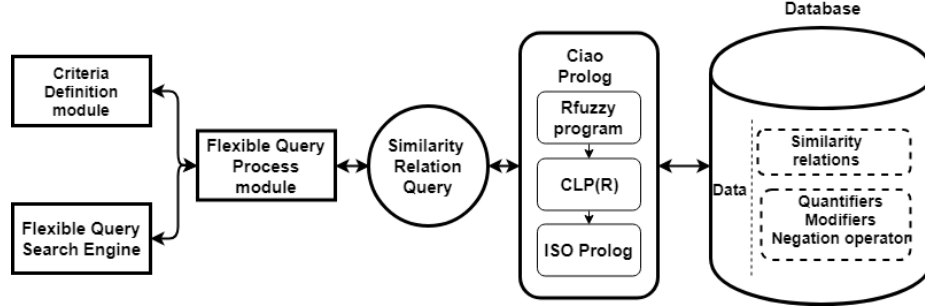


Fig. 2. System architecture

### 3.2 Data files uploading interface

We present the uploading interface in Fig. 3, in which we can see a list of data-files of various formats. The framework allows users to define the types for the existing data in the data-file. To clarify, we provide an example in Fig. 4, where we assigned the data-types for each column of the film database. After assigning the data-types, the interface will prepare the configuration file (Prolog file) which stores the similarity relation for performing fuzzy and flexible searches. We present the structure of the configuration file in (Example 3.1).

*Exapmle 3.1* We take an example of a film database that by the way was in SQL format, which contains 6 columns and 16 records. After uploading it to the system (Fig. 3) and assigning the data-types (Fig. 4), the database gets converted into a configuration file (Prolog). The contents of the configuration file are as follows:

```

% Configuration file (Prolog file):
:- module(car, _, [rfuzzy, clpr]).






% database definition define_database(film/6,
[(film_name, rfuzzy_string_type),
(release_year, rfuzzy_integer_type),
(duration_in_minutes,
rfuzzy_integer_type),

```

```

(genre, rfuzzy_enum_type),
(original_language, rfuzzy_enum_type),
(directed_by, rfuzzy_enum_type))).
film('The Godfather',1972, 207, drama, english, 'Francis Ford Coppola').
film('Casablanca', 1946, 172, romance, english, 'Michael Curtiz').
film('Cera una volta il West',1968,165,western,italian,'Sergio Leone').
film('El laberinto del fauno',2006,107,drama,spanish,'Guillermo del Toro').
film('Il buono, il brutto, il cattivo', 1967, 141,adventure,italian, 'Sergio Leone').
film('Finding Nemo', 2003, 112, comedy, english, 'Andrew Stanton and Lee Unkrich').
film('Thor - The dark world', 2013, 90, action, english, 'Alan Taylor').
film('Blue Jasmine', 2013, 98, action, english, 'Woody Allen').
film('The Collection', 2013, 82, thriller, english, 'Marcus Dun-stan').
film('Before Sunrise', 1995, 101, romantic_drama, english, 'Rich-ard Linklater').
film('Before Midnight', 2013, 109, romantic_drama, english, 'Richard Linklater').
film('Quien mato a Bambi?', 2013, 89, comedy, spanish, 'Santi Amodeo'). film('Not
Suitable for Children', 2012, 96, romantic_comedy, english, 'Peter Templeman').
film('Alien vs Predator', 2004, 115, science_fiction, english, 'Paul W.S. Anderson'.
film('Despicable Me', 2010, 95, comedy, english, 'Pierre Coffin and Chris Renaud').
film('Despicable Me 2', 2013, 98, comedy, english, 'Pierre Coffin and Chris Renaud').

```

| UPLOAD YOUR DATA FILES |           | UPLOAD  |
|------------------------|-----------|---|
| UPLOADED DATA FILES    | PRIVACY   | REMOVE FILE   |
| cars.xls               | ✓ PUBLIC  |  |
| film.sql               | ✓ PUBLIC  |  |
| restaurant.json        | ✓ PUBLIC  |  |
| shopping.csv           | ✗ PRIVATE |  |
| students.xlsx          | ✗ PRIVATE |  |
| travels.pl             | ✓ PUBLIC  |  |

**Fig. 3.** Data files uploading interface



Select the types for the columns:

film\_name  release\_year  duration\_in\_minute  genre

original\_language  directed\_by

```

CREATE TABLE restaurant (
  film_name VARCHAR(30),
  release_year VARCHAR(11),
  duration_in_minute VARCHAR(11),
  genre VARCHAR(11),
  original_language VARCHAR(30) ,
  directed_by VARCHAR(30);

INSERT INTO film VALUES('The_Godfather', 1972, 207, 'drama', 'english', 'Francis_Ford_Coppola');
INSERT INTO film VALUES('Casablanca', 1946, 172, 'romance', 'english', 'Michael_Curtiz');
INSERT INTO film VALUES('Cera_una_volta_il_West', 1968, 165, 'western', 'italian', 'Sergio_Leone');
INSERT INTO film VALUES('El_laberinto_del_fauno', 2006, 107, 'drama', 'spanish', 'Guillermo_del_Toro');
INSERT INTO film VALUES('Il_buono_il_brutto_il_cattivo', 1967, 141, 'adventure', 'italian', 'Sergio_Leone');
INSERT INTO film VALUES('Finding_Nemo', 2003, 112, 'comedy', 'english', 'Andrew_Stanton_and_Lee_Unkrich');
INSERT INTO film VALUES('Thor_The_dark_world', 2013, 90, 'action', 'english', 'Alan_Taylor');
INSERT INTO film VALUES('Blue_Jasmine', 2013, 98, 'action', 'english', 'Woody_Allen');
INSERT INTO film VALUES('The_Collection', 2013, 82, 'thriller', 'english', 'Marcus_Dunstan');
INSERT INTO film VALUES('Before_Sunrise', 1995, 101, 'romantic_drama', 'english', 'Richard_Linklater');
INSERT INTO film VALUES('Before_Midnight', 2013, 109, 'romantic_drama', 'english', 'Richard_Linklater');
INSERT INTO film VALUES('Quien_mato_a_Bambi?', 2013, 89, 'comedy', 'spanish', 'Santi_Amodeo');
INSERT INTO film VALUES('Not_Suitable_for_Children', 2012, 96, 'romantic_comedy', 'english', 'Peter_Templeman');
INSERT INTO film VALUES('Despicable_Me', 2010, 95, 'comedy', 'english', 'Pierre_Coffin_and_Chris_Renaud');
INSERT INTO film VALUES('Despicable_Me_2', 2013, 98, 'comedy', 'english', 'Pierre_Coffin_and_Chris_Renaud');

```

Fig. 4. assigning data-types for the columns

As we can see, the configuration file is made up of four different parts: (1) the header ":- module(film,\_,[rfuzzy, clpr])." that includes the packages and libraries of RFuzzy and CLPR (Constraint Logic Programming) which helps us to introduce the structure of fuzzy criteria to the search engine while posing an expressive query. (2) is the definition of the database that defines the columns and their data-types, (3) is a table storing the crisp data. (4) Is the part where the similarity relation defined by users get stored.

### 3.3 Defining similarity relation interface

In order to perform a query based on the similarities, you need to have the criteria already defined inside the configuration file. Since the low-level syntax for defining the similarity criteria (as defined in Section 2) is not so easy for a user without having knowledge of database or programming, therefore, we present a user-friendly interface in Fig. 5 that allows end-users to create the similarity relation criteria (without being concerned about the low-level syntax of the criteria), and then they can apply that criteria over various types of modern and conventional data formats such as (JSON, SQL, Prolog, CSV, XLS, and XLSX) for performing flexible queries. Once the data file is uploaded to the system, with the help of "Define new similarity relations" option the user can create the criteria by assigning a degree of similarities between two values. We have explained the steps for defining similarity relations in below.

The steps for creating a similarity relation are as follows:

- i. Selecting the database: As there can be more than one database in a configuration file such as (film, restaurant, etc.) therefore, the interface asks the user to select the one on which he/she wants to define the similarity relation criterion.
- ii. Selecting the column: In this step, the interface asks for the column on which the users want to define the similarity relations. Such as (genre, etc.).
- iii. Selecting the values: after selecting the column, the interface asks for two values/items such as (romance, drama, action, comedy, horror, etc.) on which the user want to assign the similarity degree.
- iv. Defining the similarity degree: Finally, the user has to assign a similarity degree (a real number between 0 and 1) to define whether the items are (completely different, quite different, very different, similar, very similar, and completely similar). After clicking the save button, the criteria get created inside the configuration file.

**Fig. 5.** Defining similarity relation interface

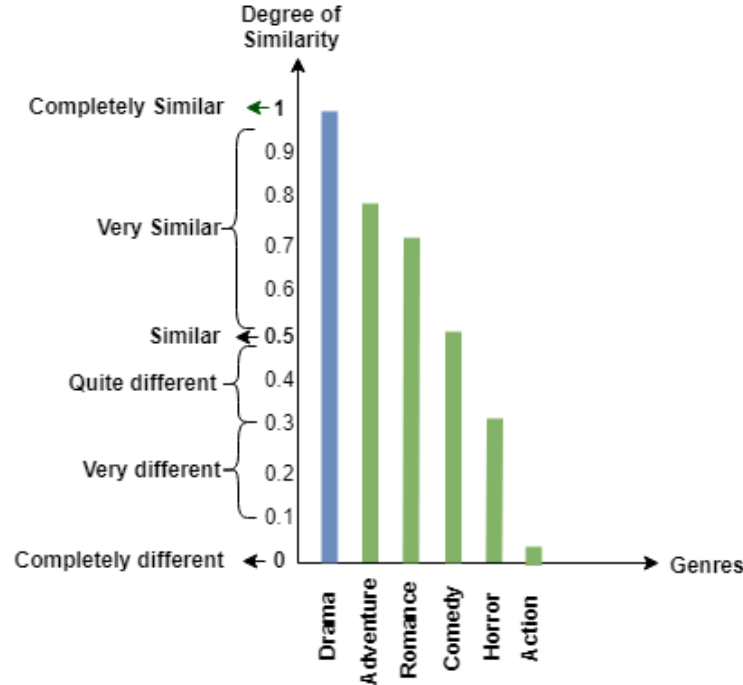
*Example 3.1 (Continued)* Once the configuration file is generated for the film database, now the user can create the similarity relations over the values of the film database. For example, the user wants to define the similarity relations between "drama" genre and other genres such as (romance, action, comedy, horror, and adventure). Thus, the similarity relations will get created as follows:

```
%Similarity relations defined over genres:
define_similarity_between(film, genre(drama), genre(adventure), 0.8).
define_similarity_between(film, genre(drama), genre(romance), 0.7).
define_similarity_between(film, genre(drama), genre(comedy), 0.5).
define_similarity_between(film, genre(drama), genre(horror), 0.3).
define_similarity_between(film, genre(drama), genre(action), 0).
```

We can see that the user has created five similarity relations between the genre "drama" and "romance, adventure, comedy, horror, and action", and the user has presented the similarity by assigning them different degree (between 0 and 1). We have considered six different categories for the similarity degrees created in our framework to make the end-users understand what we exactly mean by these real numbers (0 and 1) while defining the similarity relation. These categories are as follows:

- i. Completely different (Category 0): This category has the value "0", and the user is going to assign it for the similarity relation between the values, that he/she thinks they are "completely different".
- ii. Very different (Category 0.1 – 0.3): This category has the values between "0.1" and "0.3", the user will assign these degrees between the values that he/she thinks they are "very different".
- iii. Quite different (Category 0.3 – 0.5): This category has the values between 0.3 and 0.5, the user assigns these degrees between the values that he/she thinks they are "quite different".
- iv. Similar (Category 0.5): This category defines the relation between the values that the user thinks they are "similar" to each other.
- v. Very similar (Category 0.6 – 0.9): By assigning any degree of this category in a similarity relation, the user means the two values are "very similar" to each other.
- Completely similar (Category 1): This category has the value "1", and the user assign it for the similarity relation between the values which are "completely similar".

We present a graph in (Fig. 6) for the similarity relations that we have created in our previous example:



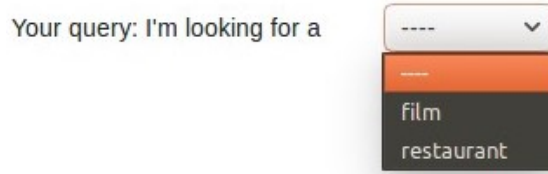
**Fig. 6.** Similarity relation between drama and other genres.

### 3.4 Query interface

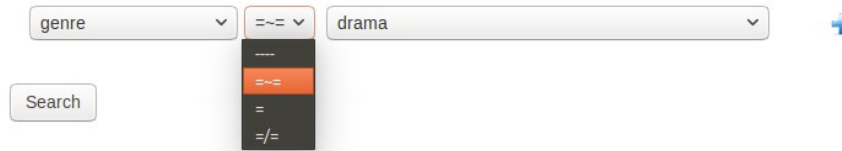
In order to perform a query over a database we need to follow the following steps:

- i. In querying interface, we select the database (such as film, restaurant, etc.) on which we want to pose the query as shown in Fig. 7.
- ii. After selecting the database, the framework will provide us a list of predicates such as (genre, film\_name, release\_year, original\_language, etc.) and their values such as (drama, horror, action, romance, comedy, etc.) as shown in Fig. 8, we need to select the predicate with an enum\_type and a corresponding value, and then we define the modifier similar ( $=$ ) between them.
- iii. After defining all the criteria, we click the search button for executing our query.

*Example 3.1 (Continued):* After defining the similarity relations between "drama" and other genres (romance, action, comedy, horror, and adventure) of the film database, now the user wants to query: "I am looking for the film whose genre is similar to drama" Or, "I am looking for the film whose genre is similar to romance, etc." By posing these queries, our system will provide us a list of all the films from different categories devoted on the similarity degrees assigned for each relation (completely different, very different, quite different, similar, very similar, and completely similar).



**Fig. 7.** Selecting the database.



**Fig. 8.** Selecting the predicates and the modifiers.

### 3.5 Answering interface

The answering interface provides different sets of in different tabs (10 best results, results in over 70%, results in over 50%, results in over 0 %, and all the results). These tabs provide different choices to the users to select the one which satisfies his/her needs.

- The tab with "10 best results" provides us the 10 best items (or less, it depends on the number of items existed in the database) which are similar to each other with a highest degree of similarity.

- The tab with the "results over 70%" provides us with a greater number of options with the similarity degrees between the range of (0.8 to 1). That means the user can get the items from the categories (very similar, and completely similar).
- The tab with the "results over 50%" provides us the items with the similarity degrees between the range of (0.6 to 1) that includes the items from the categories (similar, very similar, and completely similar).
- The tab with "the results over 0%" includes the items from the categories (quite different, very different, similar, very similar, and completely similar) with the similarity degree between the range of (0.1 to 1) excluding "0".
- And the tab with "all results" provide us the items from all the categories with a similarity degree from (0 to 1) including "0".

Ajax has been used for getting the result form each tab to avoid waste of computing time.

*Example 3.1 (Continued):* After posing the query, "I am looking for the film whose genre is similar to drama." The results provided by the system are as follows: The system provided us with the top 8 best results out of 10 (after selecting the tab with 10 best results) as there is only eight films which are having the highest similarity degree with drama films.).

| 10 best results | Results over 70%                | Results over 50% | Results over 0%     | All results |                   |                                |             |
|-----------------|---------------------------------|------------------|---------------------|-------------|-------------------|--------------------------------|-------------|
| film            | film name                       | release year     | duration in minutes | genre       | original language | directed by                    | Truth Value |
| nº.1            | The Godfather                   | 1972             | 207                 | drama       | english           | Francis Ford Coppola           | 1           |
| nº.2            | El laberinto del fauno          | 2006             | 107                 | drama       | spanish           | Guillermo del Toro             | 1           |
| nº.3            | Il buono, il brutto, il cattivo | 1967             | 141                 | adventure   | italian           | Sergio Leone                   | 0.8         |
| nº.4            | Casablanca                      | 1946             | 172                 | romance     | english           | Michael Curtiz                 | 0.7         |
| nº.5            | Finding Nemo                    | 2003             | 112                 | comedy      | english           | Andrew Stanton and Lee Unkrich | 0.5         |
| nº.6            | Quien mato a Bambi?             | 2013             | 89                  | comedy      | spanish           | Santi Amodeo                   | 0.5         |
| nº.7            | Despicable Me                   | 2010             | 95                  | comedy      | english           | Pierre Coffin and Chris Renaud | 0.5         |
| nº.8            | Despicable Me 2                 | 2013             | 98                  | comedy      | english           | Pierre Coffin and Chris Renaud | 0.5         |

**Fig. 9.** The 10 best results tap.

We can see in the results (Fig. 9, it includes two films of genre "drama" with a degree "1", which indicates that all the films from the same genre (for example drama and drama) are "completely similar" to each other. The third best answer is the film with the genre "adventure" with a similarity degree "0.8", that indicates this film is "very similar" to the drama film based on the relation assigned between them. The fourth best answer is the "romance" film with a degree "0.7", and the rest four movies are with degree "0.5" which are "similar" to "drama" film. We can see that, with the help of the similarity degree assigned to each film a user can easily conclude which film is "similar, very similar, and completely similar" to "drama".

## 4 Conclusions

We have presented a framework which is a flexible and fuzzy search engine for querying over various data formats devoted on similarity relations. We provide details about the syntax and semantics of the constructions for representing the real-world similarity relations using fuzzy logic (Section 2). We have presented the details about the implementation and the architecture of our system (Section 3) with a comprehensive introduction about the steps for creating and modifying the criteria, and we have described the querying structure of our framework and the results to justify our work. The distinctive characteristics of our system in comparison to the existing ones are: (1) we do not force the similarity relation to be reflexive, symmetric and transitive, i.e., an equivalence relation. (2) the promising prototypes are devoted to minority data formats as Prolog, but our framework provides the possibilities to the user to pose their expressive (fuzzy) queries devoted to similarity relations over various conventional and modern data formats. (3) our user-friendly interface that reduces that gap between the database users and the logic programming users by allowing them to search their crisp data in conventional formats flexibly and expressively without knowing about the low-level syntax and semantics of the fuzzy relations. As a result, we obtain a fuzzy search engine for querying over modern and conventional data formats devoted to the similarity relations between the real-world concepts. We believe, our work contributes to the advancement of intelligent search engines. Our current research is on allowing users to personalize fuzzy criteria for their flexible searches over databases. so that every user of our system will be able to pose queries and get the result based on his/her personalized criteria.

## References

1. Baldwin, J.F., Martin, T.P., Pilsworth, B.W.: *Fril- Fuzzy and Evidential Reasoning in Artificial Intelligence*. John Wiley & Sons, Inc., New York, NY, USA (1995)
2. Bobillo, F., Straccia, U.: fuzzyDL: An expressive fuzzy description logic reasoner. In: *International Conference on Fuzzy Systems (FUZZ-08)*. pp. 923–930. IEEE Computer Society (2008)
3. Dubois, D., Prade, H.: Comparison of two fuzzy set-based logics: similarity logic and possibilistic logic. In: *Proceedings of 1995 IEEE Int. Fuzzy Systems, International Joint Conference of the Fourth IEEE International Conference on Fuzzy Systems and The Second International Fuzzy Engineering Symposium*. vol. 3, pp. 1219–1226 (1995)

4. Esteva, F., Garcia, P., Godo, L., Ruspini, E.H., Valverde, L.: On similarity logic and the generalized modus ponens. In: Proceedings of the Third IEEE Conference on Computational Intelligence, Fuzzy Systems, IEEE World Congress on Computational Intelligence. vol. 2, pp. 1423–1427 (1994)
5. Godo, L., Rodríguez, R.O.: A fuzzy modal logic for similarity reasoning. In: Cai, K.Y., Chen, G., Ying, M. (eds.) *Fuzzy Logic And Soft Computing*. Kluwer Academic (1999)
6. Guadarrama, S., Muñoz-Hernández, S., Vaucheret, C.: Fuzzy prolog: a new approach using soft constraints propagation. *Fuzzy Set. Syst.* **144**(1), 127–150 (2004)
7. Ishizuka, M., Kanai, N.: Prolog-elf incorporating fuzzy logic. In: Proceedings of the 9th international joint conference on Artificial intelligence. pp. 701–703. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA (1985)
8. Li, D., Liu, D.: *A Fuzzy Prolog Database System*. John Wiley & Sons, Inc., New York, NY, USA (1990)
9. Lloyd, J.W.: *Foundations of logic programming*. Springer-Verlag, Berlin (1987), Second edition
10. Medina, J., Ojeda-Aciego, M., Vojtáš, P.: Similarity-based unification: a multi-adjoint approach. *Fuzzy Set. Syst.* **146**(1), 43–62 (2004)
11. Medina, J., Ojeda-Aciego, M., Vojtáš, P.: A multi-adjoint approach to similarity-based unification. *Electr. Notes Theor. Comput. Sci.* **66**, 70–85 (2002)
12. Morcillo, P., Moreno, G.: Floper, a fuzzy logic programming environment for research. In: Gij (ed.) *Proceedings of VIII Jornadas sobre Programacion y Lenguajes (PROLE'08)*. pp. 259–263 (10 2008)
13. Muñoz-Hernández, S., Pablos-Ceruelo, V., Strass, H.: Rfuzzy: Syntax, semantics and implementation details of a simple and expressive fuzzy tool over prolog. *Inf. Sci.* **181**(10), 1951–1970 (2011)
14. Pablos-Ceruelo, V., Muñoz-Hernández, S.: Introducing priorities in rfuzzy: Syntax and semantics. In: *CMMSE 2011 : Proceedings of the 11th International Conference on Mathematical Methods in Science and Engineering*. vol. 3, pp. 918–929. Benidorm (Alicante), Spain (6 2011)
15. Pablos-Ceruelo, V., Muñoz-Hernández, S.: Getting answers to fuzzy and flexible searches by easy modeling of real-world knowledge. In: *IJCCI 2013 – Proceedings of the 5th International Joint Conference on Computational Intelligence* (2013)
16. The CLIP lab: "The Ciao Prolog Development System", <http://www.clip.dia.fi.upm.es/Software/Ciao>
17. Vojtáš, P.: Fuzzy logic programming,. *Fuzzy Set. Syst.* **124**(3), 361–370 (2001)
18. Wang, J.B., Xu, Z.Q., Wang, N.C.: A fuzzy logic with similarity. In: *Proceedings of the 2002 International Conference on Machine Learning and Cybernetics*. vol. 3, pp. 1178–1183 (2002)
19. Zadeh, L.A.: Fuzzy sets. *Inf. and Control* **8**(3), 338–353 (1965)
20. Zadeh, L.A.: Calculus of fuzzy restrictions. In: *Fuzzy Sets and Their Applications to Cognitive and Decision Processes*. pp. 1–39 (1974)