BOUSI∼PROLOG:
A Fuzzy Logic Programming Language for Modeling Vague Knowledge and Approximate Reasoning

Pascual Julián Iranzo, Clemente Rubio Manzano
Department of Information Technologies and Systems, University of Castilla-La Mancha, Ciudad Real, Spain
Pascual.Julian@uclm.es,Clemente.Rubio@uclm.es

Keywords: Fuzzy Logic Programming, Fuzzy Prolog, Weak SLD Resolution, Bousi∼Prolog Applications

Abstract: Bousi∼Prolog is an extension of the standard Prolog language. Its operational semantics is an adaptation of the SLD resolution principle where classical unification has been replaced by a fuzzy unification algorithm based on fuzzy relations defined on a syntactic domain. In this paper we describe how Bousi∼Prolog may contribute to resolve several problems extracted from different application areas, where it is mandatory to deal with vagueness and uncertain knowledge, such as: flexible deductive databases, fuzzy control, fuzzy experts systems, data retrieval or approximate reasoning. Hence, through several (small but meaningful) examples we show the great potential of this programming language.

1. Introduction

Logic Programming has been used widely in the area of Artificial Intelligence for knowledge representation, expert system development or design of deductive databases. However, in recent years, it has lost ground in the Artificial Intelligence scene. One reason, perhaps the most important, has been the inability of Logic Programming languages to represent and to handle in an explicit, natural way the vagueness and/or the imprecision existing in the real world. Vagueness and/or imprecision involve concepts that cannot be sharply defined, such as: tall, small, close, far, cheap, expensive, etc. Our main concern is representing, manipulating, and drawing inferences from such imprecise notions. Certainly, dealing with approximate reasoning and vagueness is essential in most Artificial Intelligence application areas, such as expert systems, fuzzy control, robotics, computer vision, machine learning or information retrieval. So, it is crucial to enhance these languages with new facilities. If we want to preserve them as useful programming tools for Artificial Intelligence, it is necessary to allow an explicit management of vagueness and approximate reasoning.

In order to address these limitations, the Fuzzy Logic Programming (FLP) paradigm was born, so early as in the seventies of the past century. FLP is a research area which investigates how to introduce fuzzy logic concepts into logic programming aiming at to deal with the imprecision and/or vagueness existing in the real world. Despite it has long ancestry, it is not a well established area with firm standards. There are multiple lines of work between the one that modifies the classical resolution procedure, replacing it by a fuzzy resolution mechanism (Lee (1972); Li and Liu (1990); P. Vojtáš (2001)) and the one that extends the classical SLD resolution principle, modifying its classical unification algorithm and replacing it by a fuzzy unification algorithm (Arcelli and Formato (2002); Sessa (2002); Virtanen (1998)).

One distinguished feature of the first approach of FLP is that it has produced programming languages with explicitly annotated rules. These annotations or weights are expressing the confidence which the user of the system has in the truth of the rules integrating a program. This way of expressing uncertainty may be inappropriate, or at least inconvenient, in some contexts. Also it seems unnatural, from a declarative point of view, to specify vagueness by means of a precise number. If we want to be more accurate to the spirit of declarative programming, we should find mechanisms that allow us to express the vagueness in a way independent from the program. That is, surpass the well known assertion stating that an Algorithm = Logic + Control (Kowalski (1979)) and to move to a new assertion: an Algorithm = Logic + Vague Knowledge + Control.

Bousi∼Prolog (Julián and Rubio (2009); Julián and Rubio (2009); P. Julián, et al (2009)) is a FLP language, that may be classified among those programming languages pertaining to the second approach mentioned above, whose design has been conceived to make a clean separation between Logic, Vague Knowledge and Control. In a Bousi∼Prolog program Logic is specified by a set of Prolog facts and rules, Vague Knowledge is mainly specified by a set of (what we call) *proximity equations*, defining a fuzzy binary relation (which is expressing how close are two concepts), and Control is let automatic to the system, through an enhanced SLD resolution procedure with a fuzzy unification algorithm based on fuzzy binary relations. Informally, this *weak* unification algorithm states that two terms $f(t_1,...,t_n)$ and $g(s_1,...,s_n)$ weakly unify if the root symbols f and g are close, with a certain degree, and each of their arguments t_i and s_i weakly unify. Therefore, the weak unification algorithm does not produce a failure if there is a clash of two syntactical distinct symbols, whenever they are approximate, but a success with a certain approximation degree. Hence, Bousi∼Prolog computes answers as well as approximation degrees. These features are evidenced by the following example.

Assume a fragment of a deductive database that stores information about people and their preferences. Under the question about whether somebody likes whiskey, that person may answer: 'I do not like', 'a little', 'so much', 'very much'. This situation can be modeled by a Bousi∼Prolog program, where vagueness and preferences of users are specified by a set of proximity equations.

```
%% PROXIMITY EQUATIONS
does_not~a_little=0.5. a_little~very_much=0.2.
does_not~so_much=0.1.  so_much~very_much=0.6.
a_little~so_much=0.4.

%% FACTS
likes(john, whiskey, a_little).
likes(mary, whiskey, very_much).
likes(peter, whiskey, so_much).
likes(paul, whiskey, does_not).

%% RULES
buy(X,P):-likes(X, P, very_much).
```

In a standard Prolog system, if we ask about people who will buy scotch whiskey ``?-buy(X,whiskey)'', only one answer is obtained: Yes, X=mary. However john and peter also are reasonable candidates to buy scotch whiskey. Hence, if we are looking for a flexible query answering procedure, more accurate to the real world behavior, john and peter should be appear as answers. The Bousi∼Prolog system answers ``X=john with 0.2'', ``X=mary with 1.0'', and ``X=peter with 0.6''. To obtain the first answer, the Bousi∼Prolog system operates as follows: since we have specified that a_little is close to very_much, with degree 0.2, these two terms may unify "weakly" with approximation degree 0.2, leading to the unification of likes(john, whiskey, a_little) and likes(X, whiskey, very_much), with X=john and approximation degree 0.2; therefore, the assertion buy(john,whiskey) is stated with approximation degree 0.2. Similarly for the remainder answers.

The aim of this paper is to describe how Bousi∼Prolog may contribute to resolve several problems extracted from different application areas, where it is mandatory to deal with vagueness and uncertain knowledge, such as: deductive databases, knowledge-based systems, data retrieval or approximate reasoning. To accomplish this goal, through the paper, we discuss and implement several (small but meaningful) examples showing the great potential of this programming language. For a more detailed study of Bousi∼Prolog foundations, as well as pieces of related work, we direct the interested reader to the papers (Julián and Rubio (2009); Julián and Rubio (2009); P. Julián, et al (2009)). Also, a reference manual (with its precise syntax) and more examples can be found at the URL: http://dectau.uclm.es/bousi/.

2. Flexible Query Answering in Deductive Databases

There are several approaches to fuzzy flexible database. We highlight two of them: i) the model of Buckles-Petry (Buckles and Petry (1985)) and Shenoi-Melton (Shenoi and Melton (1999)), where a relational database is fuzzified by means of a similarity/proximity relation defined on an universal domain; ii) the model of Prade-Testemale (Prade and Testemale (1984)) in which attribute values may be expressed in (vague) linguistic terms (roughly speaking, using fuzzy sets).

In this section we show how Bousi∼Prolog allows to simulate both fuzzy flexible database approaches effectively. We begin with an example illustrating how to implement a fragment of a flexible database in the the Buckles-Petry and Shenoi-Melton style. This example is adapted from a work of Motro (Motro (1988)), although the techniques used there differs from ours. In Motro's work the relational data model is extended using *data metrics*, and the query language is extended with a single feature, a *similar-to comparator*. On the other hand, we concentrate on deductive databases, i.e., databases consisting of facts and rules that have a deductive capability.

Assume a database storing information on films which are shown at some cinema of a specific neighbourhood of Los Angeles city. The database consists of three tables (relations, represented by BPL facts) with a total of eight attributes. The `film` table has three attributes: title, director and category of the film. The `theater` table is characterized by the theater name, owner and location of the theater. The `engagement` table is used to link the information stored in the first two tables and it has two attributes: the title of the film and the name of the theater. The fuzzy component is defined by two proximity relations. The first one states the similarity between the different film categories (i.e., it is defined on the syntactic domain of film categories) and the second one states the closeness of two theater locations (i.e., it is defined on the syntactic domain of theater locations). In this example, both fuzzy relations are implemented explicitly by means of a set of proximity equations.

```
%% DIRECTIVE
:-lambdaCut(0.5).  %% only approximation degrees equal or
                   %% greater than 0.5 are considered
%% PROXIMITY RELATIONS
%% Location Distance Relationship
bervely_hills~downtown=0.3.      downtown~santa_monica=0.23.
bervely_hills~santa_monica=0.45. downtown~westwood=0.25.
bervely_hills~hollywood=0.56.    hollywood~santa_monica=0.3.
bervely_hills~westwood=0.9.      hollywood~westwood=0.45.
downtown~hollywood=0.45.         santa_monica~westwood=0.9.

%% Category Relationship
comedy~drama=0.6.                drama~adventure=0.6.
comedy~adventure=0.3.            drama~suspense=0.6.
comedy~suspense=0.3.             adventure~suspense=0.9.

%% Films Table
%% film(Title, Director, Category)
film(four_feathers, korda, adventure).
film(modern_times, chaplin, comedy).
film(psycho, hitchcock, suspense).
film(rear_window, hitchcock, suspense).
film(robbery, yates, suspense).
film(star_wars, lucas, adventure).
film(surf_party, dexter, drama).

%% Theaters Table
%% theater(Name,Owner,Location).
theater(chinese, mann, hollywood).
theater(egyptian, va, westwood).
theater(music_hall, lae, bervely_hills).
theater(odeon, cineplex, santa_monica).
theater(rialto, independent, downtown).
theater(village, mann, westwood).

%% Engagements Table
%% engagement(Film,Theater)
engagement(modern_times, rialto).
engagement(start_wars, rialto).
engagement(star_wars, chinese).
engagement(rear_window, egyptian).
engagement(surf_party, village).
engagement(robbery, odeon).
engagement(modern_times, odeon).
engagement(four_feathers, music_hall).

%% MAIN RULE
%% search(input, input, output, output)
search(Category, Location, Film, Theater) :-
      film(Film, _, Category), engagement(Film, Theater),
      theater(Theater, _, Location).
```

The predicate `search/4` allows us to know the cinema which is showing a film closest to our location and category of preference. If we launch the goal "`search(adventure, westwood, Film, Theater).`", the system answers: "`Film = rear_window, Theater = egyptian, with 0.9`" (a suspense film located at `west- wood`), "`Film = surf_party, Theater = village, with 0.6`" (a drama film located at `westwood`), "`Film = robbery, Theater = odeon, with 0.9`" (a suspense film located at `santa_monica`), "`Film = four_feathers, Theater = music_hall, with 0.9`" (an adventure film located at `bervely_hills`).

Next, we present a BPL program implementing a fragment of a flexible deductive database in the style of Prade and Testemale. That is, databases that incorporate the notion of fuzziness by means of fuzzy sets that may be used as attributes of a table.

In this example we model a database fragment for a real state company with information about flats to be hired. The company wants to help clients to select flats in stock, according their preferences.

```
%% DIRECTIVES declaring and defining linguistic variables
%% (i.e., fuzzy sets)
%% Linguistic variable: rental
:-domain(rental(0,600,euros)).
:-fuzzy_set(rental[cheap(100,100,250,500),
      normal(100,300,400,600), expensive(300,450,600,600)]).

%% Linguistic variable: distance
:-domain(distance(0,50,minutes)).
:-fuzzy_set(distance[close(0,0,15,40),
                 medial(15,25,30,35), far(20,35,50,50)]).

%% Linguistic variable: flat conditions
:-domain(condition(0,10,conditions)).
:-fuzzy_set(conditions[unfair(0,0,1,3), fair(1,3,6),
                    good(4,6,8), excellent(7,9,10,10)]).

%% FACTS
%% Flats table
%% flat(Code, Street, Rental, Condition).
flat(f1, libertad_street, rental#300, more_or_less#good).
flat(f2, ciruela_street, rental#450, somewhat#good).
flat(f3, granja_street, rental#200, unfair).
```

```
%% Streets table
%% street(Name, District)
street(libertad_street, ronda_la_mata).
street(ciruela_street, downtwon).
street(granja_street, ronda_santa_maria).
%% Distance table
%% distance(District, District, Distance)
%% to university campus
distance(ronda_la_mata, campus, somewhat#close).
distance(downtwon, campus, medial).
distance(ronda_santa_maria, campus, far).

%% RULES
%% close_to(Flat, District)
close_to(Flat, District):-
     flat(Flat, Street, _, _), street(Street, Flat_Dist),
     distance(Flat_Dist, District, close).

select_flat(Flat, Street):-
     flat(Flat, Street, cheap, good), close_to(Flat,campus).
```

Now `select_flat/2` selects those flats which may be considered `cheap`, `good` and `close` to the `campus` with a certain degree. More precisely, if we launch the goal "`?- select_flat(Flat, Street).`", we obtain: "`Flat = f1, Street = libertad, with 0.8`" and "`Flat = f2, Street = ciruela, with 0.2`", being flat `f3` at `granja` street disregarded.

Finally, it is noteworthy that in many cases it will be necessary, or preferable, to combine both approaches, in order to model a problem properly. That is, to declare and define linguistic variables in conjunction with proximity relations. This last option is completely feasible in Bousi∼Prolog.

3. Knowledge-Based Systems

As an example of Knowledge-Based System, we implement a fuzzy logic controller which is an adaptation of the one described in (Shalfield (2005)) [1].

In the general model, the device being controlled has a set of activators that take the input and use this to affect its settings and a set of sensors that get information from the device. The fuzzy logic controller takes the 'crisp' information from the sensors as input and fuzzifies this into some linguistic variables, propagates membership values using linguistic rules and, finally, defuzzifies the output and returns these 'crisp' values to the activators. The controller we shall implement will use a set of fuzzy rules to adjust the throttle of a steam turbine according to its current temperature and pressure to keep it running smoothly.

[1]We suggest to compare our solution with the one appeared in (Shalfield (2005)), in order to observe how natural is our codification of this problem.

Defining linguistic variables First we need to decide upon the linguistic variables that are going to be used in the BPL program. We do this by looking at the descriptors that will be used by the rules, in this case 'temperature', 'pressure' and 'throttle':

```
:-domain(temperature(0,500,Celsius)).
:-fuzzy_set(temperature, [cold(0,0,110,165),
     cool(110,165,220), normal(165,220,275),
     warm(220,275,330), hot(275,330,500,500)]).

:-domain(pressure(0,300,kpa)).
:-fuzzy_set(pressure, [weak(0,0,10,70), low(10,70,130),
      ok(70,130,190), strong(130,190,250),
      high(190,250,300,300)]).

:-domain(throttle(-60,60,rpm)).
:-fuzzy_set(throttle, [neg_large(-60,-60,-45,-30),
     neg_medium(-45,-30,-15), neg_small(-30,-15,0),
     zero(-15,0,15), pos_small(0,15,30),
     pos_medium(15,30,45), pos_large(30,45,60,60)]).
```

For the 'throttle' variable, a negative value indicates that the throttle should be moved back and a positive value that it should be moved forward. The problem results in moving the throttle by large, medium or small amounts in the negative and positive directions.

Defining rules After defining the linguistic variables used by our program, we can proceed to define the rules. In this example the rules will cover all the possible combinations of temperature and pressure and give a resultant throttle change for each. An example of a rule specified by an expert is: If the temperature is cold and the pressure is weak then increase the throttle by a large amount. This rule has a direct translation to BPL code:

```
throttle(pos_large):- temperature(cold), pressure(weak).
```

In particular, the remainder rules for a situation of `cold` temperature and for a situation of `cool` temperature are:

```
%% Cold
throttle(pos_medium):- temperature(cold), pressure(low).
throttle(pos_small):- temperature(cold), pressure(ok).
throttle(neg_small):- temperature(cold), pressure(strong).
throttle(neg_medium):- temperature(cold), pressure(high).
%% Cool
throttle(pos_large):- temperature(cool), pressure(weak).
throttle(pos_medium):- temperature(cool), pressure(low).
throttle(zero):- temperature(cool), pressure(ok).
throttle(neg_medium):- temperature(cool), pressure(strong).
throttle(neg_medium):- temperature(cool), pressure(high).
```

We can define the rest of rules (for `normal`, `warm`, and `hot` temperature), in a similar *linguistic* declarative way.

The inputs to the logical system should be taken from the sensors of the real device, but in this simple example they are modeled by facts:

```
%% Facts
temperature(temperature#300).    pressure(pressure#150).
```

Running this BPL program, it is possible to return a 'crisp' value for the throttle by means of the BPL built-in predicate `defuzzify` [2]. If we launch the goal "`?-defuzzify(throttle(_),Y).`" we obtain the answer "`Y = throttle#-14.09`".

4. Information Retrieval.

The following example shows how proximity equations can be used as a fuzzy model for information retrieval where textual information is selected or analyzed using an ontology (Gruber (1995)), that is, a structured collection of terms that formally defines the relations among them. Hence, inside a semantic context instead of a purely syntactic context. A practical textual inference task is finding concepts which are structurally analogous. Proximity equations can help in this task. The set of proximity equations shown below has been obtained using ConceptNet (Liu and Singh (2004)), a freely available commonsense knowledge-base and natural-language-processing toolkit[3]. ConceptNet is structured as a network of semi-structured natural language fragments. It has a `GetAnalogousConcepts()` function that returns a list of structurally analogous concepts given a source concept. In our case the source concept is "wheat". The degree of structural analogy between terms is also provided by ConceptNet. The set of proximity equations is a partial view of the original output, which was hand-made adapted by ourselves.

We want to extract information on terms structurally analogous to "wheat" on a given text. In our example, the input text is borrowed from Reuters, a test collection for text categorization research[4]. The data in this collection was originally collected and labeled by Carnegie Group, Inc. and Reuters, Ltd. in the course of developing the CONSTRUE text categorization system. The fragment text of our example is one classified with the label (topic) "wheat" and processed by erasing stop words and the endings of a word stem.

```
% DIRECTIVE
:- transitivity(yes).  %% builds a similarity starting from
                       %% the proximity equations
% PROXIMITY EQUATIONS
wheat~bean=0.315.    wheat~human=0.205.   bean~crop=0.315.
wheat~corn=0.315.    bean~animal=0.35.    bean~child=0.33.
wheat~grass=0.315.   bean~corn=0.48.      bean~grass=0.315.
wheat~horse=0.315.   bean~flower=0.315.   bean~potato=0.5.
bean~horse=0.335.    bean~table=0.35.
```

```
% FACTS and RULES
%% searchTerm(T,L1,L2), true if T is a (constant) term, L1
%% is a list of (constant) terms (representing a text) and
%% L2 is a list of triples t(X,N,D); where X is a term
%% similar to T with degree D, which occurs N times in the
%% text L1
searchTerm(T,[],[]).
searchTerm(T,[X|R],L):-
      T~X=AD, !, searchTerm(T,R,L1), insert(t(X,1,AD),L1,L).
searchTerm(T,[X|R],L):- searchTerm(T,R,L).

insert(t(T,N,D), [], [t(T,N,D)]).
insert(t(T1,N1,D), [t(T2,N2,_)|R],[t(T1,N,D)|R]) :-
        T1 == T2, N is N1+N2.
insert(t(T1,N1,D), [t(T2,N2,D2)|R2],[t(T2,N2,D2)|R]) :-
        T1 \== T2, insert(t(T1,N1,D), R2, R).

% GOAL
g(T,L):-searchTerm(T, [agriculture,department,report,farm,
          own,reserve,national,average,price,loan,release,
          price,reserves,matured,bean,grain,enter,corn,
          sorghum,rates,bean,potato], L).
```

A simple session with the BPL system, launching the goal "`?- g(corn,L).`" provides the answer: "`L = [t(potato, 1, 0.48), t(bean, 2, 0.48), t(corn, 1, 1.0)] with 1.0`". The information returned by the goal can be used lately to analyze the input text or to obtain a degree of preference in a retrieval process.

It is noteworthy, that using similar techniques to the above decribeded here, in (Romero et al (2010)) we have presented a declarative approach to text categorization using the ability of Bousi∼Prolog for flexible matching and knowledge representation by means of an ontology of terms modeled by a set of proximity equations.

5. Approximate Reasoning

Approximate reasoning is basically the inference of an imprecise conclusion from imprecise premises. All the examples discussed along this paper concerns with approximate reasoning in some degree. However, in this section we want to make a reflexion on fuzzy inference, such as it was formalized by Zadeh (Zadeh (1975a); Zadeh (1975b)), and how Bousi∼Prolog can deal with this kind of reasoning.

In this approach, each granule of knowledge is represented by a fuzzy set or a fuzzy relation on the appropriate universe. The premises of an argument are expressed as fuzzy rules and a *fuzzy inference* is a generalization of *modus ponens* that can be formalized as: "`if` x `is` F `then` y `is` G" and "x `is` F'" then "y `is` G'". Roughly speaking, x and y are variables that takes values on ordinary sets U and W, F and F' are fuzzy subsets on U whilst G and G' are fuzzy subsets

[2]As a reasonable starting point we are using the standard method of "*average of the maximum*" as defuzzifying method.

[3]Available at `http://conceptnet.media.mit.edu/`.

[4]Available at `http://www.daviddlewis.com/resources/testcollections/reuters21578/`.

on W. There has been proposed several methods to compute G', though there is not consensus as to which is the best. The method proposed by Zadeh consists of identifying from F and G a fuzzy relation, R on U and W, which has a consequence over G' on W.

Bousi∼Prolog proceeds in a different way. It constructs a fuzzy relation over the fuzzy domains (more accurately, the set of linguistic labels in a linguistic variable) involved in a BPL program. This fuzzy relation is built at compile time. Afterwards, at run time, it is used by the weak SLD resolution procedure to infer an answer to a query. For instance, Bousi∼Prolog models the following fuzzy inference in a very natural way: "`if` x `is young then` x `is fast`" and "`Robert is middle`" then "`Robert is somewhat fast`".

```
:-domain(age(0,100,years)).
:-fuzzy_set(age,[young(0,0,30,50),
    middle(20,40,60,80), old(50,80,100,100)]).

:-domain(speed(0,40,km/h)).
:-fuzzy_set(speed,[slow(0,0,15,20),
    normal(15,20,25,40), fast(25,30,40,40)]).

speed(X, fast) :- age(X, young).  age(robert, middle).
```

Now, if we launch the goal "`?- speed(robert, somewhat#fast)`", the BPL system answers "`Yes with 0.375`".

On the other hand, in our case, Bousi∼Prolog may treat linguistic terms as function or predicate symbols and no limitations are imposed to the first order syntax of the language. This feature may produce big benefices with little effort.

6. Conclusions

Bousi∼Prolog is an extension of the standard Prolog language with a fuzzy unification algorithm based on proximity relations. In this paper, through a number of (small but meaningful) examples extracted from different application areas (such as deductive databases, knowledge-based systems, data retrieval or approximate reasoning), we have shown the potential power of Bousi∼Prolog for these areas and how it is an useful tool for dealing with approximate reasoning and modeling vagueness. It is noteworthy that Bousi∼Prolog, thanks to its operational machanism, employs very few extensions to the Prolog syntax, obtaining a natural codification for vague problems that can be understood easily by a Prolog programmer. This is to remark that, the main difference between the BPL system and a standard Prolog system is in the inside, not in the surface.

REFERENCES

B. Buckles and F. Petry. A fuzzy model for relational databases. *Fuzzy Sets and Systems*, 7:213–226 (1985).

F. Arcelli and F. Formato. A similarity-based resolution rule. *Int. J. Intell. Syst.*, 17(9):853–872 (2002).

T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal Human-Computer Studies*, 43(5-6):907–928 (1995).

H.Liu and P. Singh. Conceptnet – a practical commonsense reasoning tool-kit. *BT Technology Journal*, 22(4):211–226 (2004).

P. Julián and C. Rubio. A Declarative Semantics for Bousi∼Prolog. In: *Proc. of 11th Intl. Symposium on PPDP'09*. ACM SIGPLAN (2009).

P. Julián, C. Rubio and J. Gallardo. Bousi∼Prolog: a Prolog extension language for flexible query answering. In: ENTCS, vol 248, pp. 131-147. Elsevier (2009).

P. Julián and C. Rubio. A similarity-based WAM for Bousi∼Prolog. In: LNCS, vol 5517, pp. 245–252. Springer (2009).

R.A. Kowalski. Algorithm = Logic + Control. *Communications of the ACM*, 22(7):424–436 (1979).

R.C.T. Lee. Fuzzy Logic and the Resolution Principle. *Journal of the ACM*, 19(1):119–129 (1972).

D. Li and D. Liu. *A fuzzy Prolog database system*. John Wiley & Sons, Inc. (1990).

A. Motro. VAGUE: A user interface to relational databases that permits vague queries. *ACM Transactions on Office Information Systems*, 6(3):187–214 (1988).

H. Prade and C. Testemale. Generalizing database relational algebra for the treatment of incomplete/uncertain information and vague queries. *Information Science*, 34:115–143 (1984).

F.P. Romero, P. Julián, M. Ferreira and J. Gallardo. Una Aproximación Declarativa a la Clasificación de Documentos. In: *Proc. of ESTYLF 2010*, p.p: 211-216 (2010).

M.I. Sessa. Approximate reasoning by similarity-based sld resolution. *Theoretical Computer Science*, 275(1-2):389–426, 2002.

R. Shalfield. LPA-PROLOG: Flint reference. Technical report, Logic Programming Associates ltd (2005).

S. Shenoi and A. Melton. Proximity relations in the fuzzy relational database model. *Fuzzy Sets and Systems*, 100:51–62 (1999).

H.E. Virtanen. Linguistic Logic Programming. *Logic Programming and Soft Computing*, p.p: 91–128 (1998).

P. Vojtáš. Fuzzy Logic Programming. *Fuzzy Sets and Systems*, 124(1):361–370 (2001).

L.A. Zadeh. The Concept of a Linguistic Variable and its Applications to Approximate Reasoning I, II and III. J. of Information Sciences 8 and 9, Elsevier (1975).

L.A. Zadeh. Fuzzy Logic and Approximate Reasoning. Synthese, 30(3–4):407–428, Springer (1975).