A Fuzzy Logic Programming Environment for Recycling Facility Selection

Esra Çakir[a] and H. Ziya Ulukan
Faculty of Engineering and Technology, Department of Industrial Engineering, Galatasaray University, İstanbul, Turkey

Keywords: Fuzzy Logic Programming, Fuzzy Prolog, Bousi~Prolog, Recycling Facility Selection, Waste Collection Center Selection.

Abstract: Recycling of wastes is a crucial subject for a sustainable environment. One of the main problem in this area is the appropriate location of the collection centers and recycling facilities. These facilities can be paired according to the criteria such as: distance, cost, type of waste. In this paper, fuzzy linguistics Prolog is used to find importance weights of selection criteria and to match facilities for decision making process. Bousi~Prolog is a fuzzy Prolog that enables working with both fuzzy linguistic and linguistic tools to guide the Prolog systems towards computing with paradigm phrases that can be very helpful to the linguistic resources.

1 INTRODUCTION

Logic programming has been commonly used for information representation, expert system creation or deductive database in the field of Artificial Intelligence. It has, however, lost ground in the Artificial Intelligence scene in latest years. Because in real-life problems, expressions are not precise, they are approximate, even blurred. Fuzzy Logic Programming (FLP) is a research area which investigates how to introduce fuzzy logic concepts into logic programming aiming to deal with the imprecision and/or vagueness existing in the real world. The main objectives of this new language are to support flexible query answering, to allow the manipulation of fuzzy sets and to incorporate other features with which it is possible to easily handle imprecise information using declarative techniques.

As an extension of Prolog, the most commonly used logic programming language, was proposed. In order to preserve most of its syntactic characteristics, it will find the greatest distinctions in several particular structures and in their resolution and unification algorithms. For these purpose, the language was renamed as Bousi~Prolog (abbreviated BPL), with Bousi being the Spanish abbreviation for" Unificación BOrrosa por SImilitud."

The outline of the paper is as follows. Section II includes literature review. Logic programming language Prolog and Bousi~Prolog, which is formalized as a transition system based on a proximity-based unification relation, is presented in Section III. Waste facilities were chosen as the application area, matching and criteria weighting was made between waste collection centers and recycling facilities by using Bousi~Prolog in Section IV. Finally, we give our conclusions and future research lines in Section V.

2 LITERATUR REVIEW

Bousi~Prolog is an extension of the standard Prolog language that implements a weak unification algorithm based on proximity relations, that is, reflexive and symmetric binary fuzzy relations (Dubois et al., 1988, Fontana and Formato, 2002 Orchard, 1998). Bousi~Prolog has already been used in interesting real applications such as text cataloging (Cayrol et al., 1982), knowledge discovery (Fontana et al., 1999) and linguistic feedback in computer games (Formato et al., 2000), (Julian et al., 2009).

A similarity relation is an extension of the crisp notion of equivalence relation and it can be useful in any context where the concept of equality must be

[a] https://orcid.org/0000-0003-4134-7679

Çakir, E. and Ulukan, H.
A Fuzzy Logic Programming Environment for Recycling Facility Selection.
DOI: 10.5220/0008397503670374
In *Proceedings of the 11th International Joint Conference on Computational Intelligence (IJCCI 2019)*, pages 367-374
ISBN: 978-989-758-384-1

weakened. In (Sessa, 2002) a new modified version of the **L**inear resolution strategy with **S**election function for **D**efinite clauses (SLD resolution) is defined, which is named similarity-based SLD resolution (or Weak SLD resolution: WSLD). This operational mechanism can be seen as a variant of the SLD resolution procedure where the classical unification algorithm has been replaced by the weak unification algorithm formally described in (Sessa, 2002) (and reformulate in terms of a transition system in (Julian and Rubio, 2006)) (Julian and Rubio, 2010, Mukaidono et al., 1989).

The weak unification algorithm implemented in Bousi~Prolog is an extension of Martelli and Montanari's unification algorithm for syntactic unification (Julian et al., 2009, Julian and Rubio, 2009, Lee, 1972). Such algorithm extends the classical one by relaxing the strict true/false result of unification and replacing it by a real number in the unit interval [0,1], which indicates the approximation degree of the unification. Informally, the weak unification algorithm states that two expressions *f(u(1)…u(n))* and *g(u(1)…u(n))* weakly unify, if the root symbols *f* and *g* are approximate and each argument *u(i)* and *v(i)* weakly unify, with $i \in \{1,...,n\}$. Hence, if there is a syntactic clash between two different expressions, the weak unification algorithm does not produce a failure but a success with a certain approximation degree. Notice that, Bousi~Prolog computes substitutions as well as approximation degrees (Julian et al., 2009).

Although Bousi~Prolog weak unification algorithm generalizes the operating system provided in (Mukaidono et al., 1989) and improves the language's expressive authority, some disadvantages emerge when considering real-life information and knowledge bases. As pointed out lately in (Alsinet and Godo, 1998), the weak unification maintains strict features of the classical unification, such as the arity of the predicate and function symbols. The weak unification mechanism should therefore be relaxed as the resemblance between two phrases should be determined by the data contained in each expression and it should not rely on either the arity or subterms order.

3 METHODOLOGY

Classical Prolog is not able to represent and handle the vagueness and/or imprecision that exists in the real world in an explicit. To be sure, dealing with vagueness and /or imprecision is crucial in most application fields of Artificial Intelligence, such as expert systems, fuzzy control, robotics, computer vision, machine learning or data recovery. Thus, enhancing these languages with new equipment is essential (Julian and Rubio, 2009).

3.1 Prolog

Prolog is a fifth generation computer language family used in artificial intelligence applications. It was invented by Alain Colmerauer and his working group at the University of Marseille Aix in France in the early 1970s. It comes from the French "Programmation en Logique" word. In the early 1980s, the studies conducted in order to ensure logic can as a computer language have also intensified. The interest in the subject has increased to a great extent with the Japanese announcing the fifth generation computer project in 1981.

Prolog is a tool that helps human beings in the development of the necessary methods for defining and solving the problem with its structure suitable for logical and symbolic thinking.

Assume a fragment of a deductive database that stores information about people and their preferences. Under the question about whether somebody likes tea, that person may answer: `'I do not like'`, `'a little'`, `'so much'`, `'very much'`.

```
%% FACTS
likes(ally, tea, a_little).
likes(jane, tea, very_much).
likes(kim, tea, so_much).
likes(ashley, tea, does_not).

%% RULES
drink(X,Y):-likes(X, Y, very_much).
```

In a standard Prolog system, if we ask about people who will drink tea `''?-drink(X,tea)''`, only one answer is obtained: `Yes, X=jane.` However, `ally` and `kim` also are reasonable candidates to drink tea. Hence, if we are looking for a flexible query answering procedure, more accurate to the real world behaviour, ally and kim should be appear as answers.

3.2 The Bousi~Prolog Programming Language

Bousi~Prolog is an extension of the standard Prolog language. Its operational semantics is an adaptation of the **S**election-function driven **L**inear resolution for **D**efinite clauses. (SLD resolution) principle where classical unification has been replaced by a fuzzy unification algorithm based on proximity relations defined on a syntactic domain. Hence,the operational mechanism is a generalization of the similarity-based SLD resolution principle (Sessa, 2002).

The BPL syntax is mainly the Prolog syntax but enriched with a built-in symbol "~" used for describing proximity relations by means of what we call a "proximity equation". Proximity equations are expressions of the form: "<symbol> ~ <symbol> = <degree>." (Julian and Rubio, 2010).

The BPL language makes use of two directives to define and declare the structure of a linguistic variable (Sessa, 2002): `domain` and `fuzzy_set`. The `domain` directive allows to declare and define the universe of discourse or domain associated to a linguistic variable. The concrete syntax of this directive is:

```
:-domain(Dom_Name(n,m,Magnitude)).
```

where, `Dom_Name` is the name of the domain, `n` and `m` (with n < m) are the lower and upper bounds of the real subinterval [n,m], and `Magnitude` is the name of the unit wherein the domain elements are measured. For example, the directive "`:-domain(age(0,100,years)).`" Defines a domain with name `age`, whose values are numbers (between 0 and 100) measured in `years`. The `fuzzy_set` directive allows to declare and define a list of fuzzy subsets (which are associated to the primary terms of a linguistic variable) on a predefined domain. The concrete syntax of this directive is:

```
:-fuzzy_set(Dom_Name,[FSS_1(a1,b1,c1
[,d1]),
         ...,
                FSS_n(an,bn,cn[,dn]])).
```

Fuzzy subsets are defined by indicating their name, `FSS_1`, and membership function type. At this time, it is possible to define two types of membership functions: either a trapezoidal function, if four arguments are used for defining the fuzzy subset or a triangular function, if three arguments are used.

Once a domain and the fuzzy sets associated to the primary terms have been declared, composite terms may be generated through the following grammar:

```
<Term> ::= <Atomic_term> |
<Composite_term>

<Composite_term> ::=
<TModif>#<Atomic_term>

<TModif> ::= very | somewhat |
more_or_less | extremely
```

Consider the given example in subsection *3.1.Prolog*, as Bousi~Prolog allows to work with linguistic dictionaries, a list of similar concepts for the source concept "`adjectives`" could be obtained by using `WordNet::Similarity`. This list can be represented in Bousi~Prolog by means of a set of proximity equations.

```
%% PROXIMITY EQUATIONS
does_not~a_little=0.6.
a_little~very_much=0.3.
does_not~so_much=0.2.
so_much~very_much=0.7.
a_little~so_much=0.5.
```

The Bousi~Prolog system answers `''X=ally with 0.3''`, `''X=jane with 1.0''` and `''X=kim with 0.7''`. 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 0.3, these two terms may unify "weakly" with approximation degree 0.3, leading to the unification of `likes(ally, tea, a little)` and `likes(X, tea, very much)`, with X=ally and approximation degree 0.3; therefore, the assertion `drink(ally,tea)` is stated with approximation degree 0.3.

3.2.1 Approximate Reasoning

Approximate reasoning is basically the inference of an imprecise conclusion from imprecise premises. In this section we want to make a reflexion on fuzzy inference, such as it was formalized by Zadeh (Zadeh, 1965, Zadeh, 1975), and how Bousi~Prolog can deal with this kind of reasoning.

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 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* (Julian and Rubio, 2010).

Bousi~Prolog constructs a fuzzy relation over the fuzzy domains 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. (Sessa, 2002) For instance, Bousi~Prolog models the following fuzzy inference in a very natural way: *"if x is new then x is fast" and "car-A is middle" then "car-A is somewhat fast".*

```
   :-domain(age(0,100,years)).
   :-fuzzy_set(age,[new(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, new).
age(car-A, middle).
```

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

3.2.2 Flexible Query Answering in Reductive Database

Databases are elements of software that collect and store data (which can be retrieved, added, updated or removed by users). Because the data in the actual globe is often permeated by vagueness and inaccuracy, database systems should address this issue. They should also allow flexible retrieval of data. There are several approaches to fuzzy flexible database. In this study we show how to implement a fragment of a flexible database in the Prade-Testemale (Prade and Testemale, 1984).

Below, 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. This example shows a database fragment for a person who is familiar with pharmacies in the city. This person wants to find the pharmacy by preference (ie, close to home).

```
   %% DIRECTIVES declaring and defining
linguistic variables
   %% (i.e., fuzzy sets)
   %% Linguistic variable: distance
   :-domain(distance,0,60,minutes).
   :-fuzzy set(distance,
      [close(0,10,20,40),
       medial(25,35,45,50),
       far(40,45,50,60)]).

   %% FACTS
   %% Pharmacy table
   %% pharmacy(Name, Street).
   pharmacy(ph_1,st_1).
   pharmacy(ph_2,st_2).
   pharmacy(ph_3,st_3).
   pharmacy(ph_4,st_4).
   pharmacy(ph_5,st_5).

   %% Distance table
   %% distance(District, District,
Distance)
   %% to home
   distance(st_1, home,
somewhat#medial).
   distance(st 2, home, more or less
#close).
   distance(st 3, home, very#far).
   distance(st_4, home, close).
   distance(st 5, home, somewhat#far).

   %% RULES
   %% find(Pharmacy, Distance)
   find(Pharmacy,Street, Distance):-
pharmacy(Pharmacy, Street),
   distance(Street, home, Distance).
```

Now select `find/2` (The number "2" means there is two input for query "`find`".) selects those which may be considered close to home with a certain degree. More precisely, if we launch the goal "`?- find(Pharmacy, st_1, close).`", we obtain: "`Yes, Pharmacy = ph_1 with 0.3`". This means that there is a pharmacy named "`ph_1`" on the street named "`st_1`" and approximation degree to home is 0.3. If we launch the goal "`?- find(Pharmacy, st_4, close).`", we obtain: "`Yes, Pharmacy = ph_4 with 1.0`". This means that there is a pharmacy named "`ph_4`" on the street named "`st_4`" and approximation degree to home is 1.0.

4 APPLICATION

Assume a flexible deductive database in style of Prade and Testemale (Prade and Testemale, 1984) storing information on fourteen recycling facilities and three waste collection centers which have properties such as cost, distance and energy generation capacity. These properties are expressed in fuzzy numbers. The Table 1 shows the fuzzy expressions.

Table 1: Fuzzy Expressions of Criteria.

Cost	**cheap**	**normal**	**expensive**
	(200,200,450,600)	(300,450,650,700)	(550,750,800,900)
Distance	**close**	**medial**	**far**
	(0,0,30,80)	(20,45,70,90)	(50,85,110,120)
Energy Generation Capacity	**fair**	**good**	**excellent**
	(0,0,1,3)	(2,5,7)	(6,8,9,10)

Cities where recycling facilities are located are located in different places. The recycling costs of these facilities are known. However, the energy generation capacity of the facility is expressed by using fuzzy numbers instead of crisp numbers for reasons such as material and machine destruction. Similarly, the distance of waste collection centers to the zones is expressed in fuzzy numbers. These relationships are shown in the Figure 1.

Our aim is to match waste collection center and recycling facility according to cost, distance and capacity. A BPL program code is prepared in APPENDIX. Here is the query used to run the code.

```
   %% GOAL EXAMPLE

   % find (Facility, City, Collection,
Cost ,Distance , Capacity).
```

Now "`find/6`" selects those facilities, which may be considered "`cheap`", "`close`" and "`excellent`" to the "`collection_no`" with a certain degree. More precisely:

- If we launch the goal "`?- find(Facility, City, Collection, Cost ,Distance , Capacity).`", the answer is: "`Facility = facility_1, City = city_A, Collection = collection_1, Cost = cost_400, Distance = medial, Capacity = fair, With approximation degree: 1.0`". This means that the first match with the `approximation degree = 1.0` and this is the properties of the match.

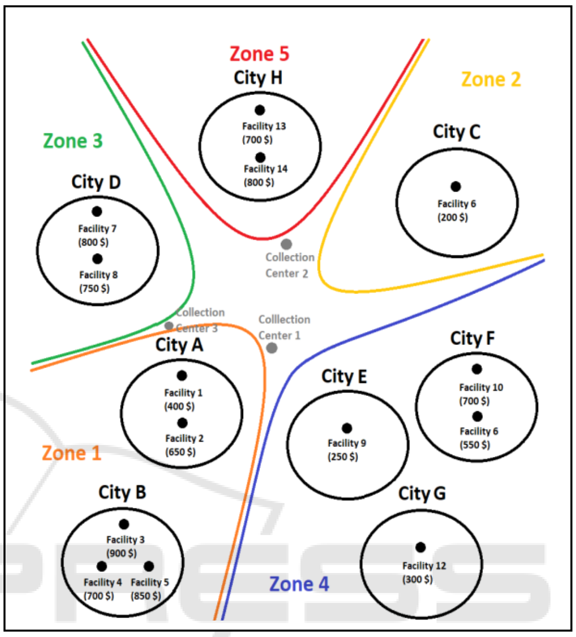

Figure 1: Facility locations and recycling costs.

- If we write six criteria to our query, the query will give us the degree of membership of this match. For example, if we launch the goal "`?-find (facility_13, city_H, collection_2, normal ,close , good).`", the answer is: "`Yes. With approximation degree: 0.92`". Because we asked the code; `Compare facility_13 in city_H and collection_2 according to cheapness, closeness and good energy capacity.`
- Also, we can find the best match of the degree of membership according to the selected criteria. For example, if we launch the goal "`?-find (Facility, City, collection_3, expensive, medial, excellent.`", the answer is: "`Facility = facility_7, City = city_D, With approximation degree: 0.82`". Because we asked the code; `find a Facility for match with collection_3 according to expensive cost, medial distance`

`and excellent energy capacity.` So, the answer is "`Yes, i found facility_7 to match with approximation degree: 0.82″.`

- These queries are based on criteria. In the last question, if we make the distance "`close″` instead of "`medial″`, the approximation will be different even if it matches the same facility. For example; "`?-find(Facility, City, collection_3, expensive , close , excellent.`", the answer is: "`Facility = facility_7, City = city_D, With approximation degree: 0.517″.` Because we asked the code; `find a Facility for match with collection_3 according to expensive cost, close distance and excellent energy capacity.` So, the answer is "`Yes, i found facility_7 to match with approximation degree: 0.517″.`
- If an irrelevant match is asked, the code answers "`No answers″.` This means that the approximation is zero. For example; "`?-find(Facility,City,collection_1, expensive , far , fair.`", the answer is: "`No answers″.` Because we asked the code; `find a Facility for match with collection_1 according to expensive cost, far distance and fair energy capacity.` There are no suitable facilities for a match in these properties.
- As a result, if we want to match waste collection centers `(collection_(1-2-3))` and recycling facilities `(facility_(1…14))` that meet the criteria of `"cheap"`, `"close"` and `"excellent"`, our query should be `"?-find (Facility, City, collection_ (1-2-3), cheap, close, excellent).″`. The best match is `″collection_1″` and "`facility_11″` with the approximation degree : 0.277. The outputs are shown in Table 2.

The advantage of using this program is to estimate the weight of these criteria. Thanks to these queries, weighting can change according to the changes in criteria. An example is given below.

Example: Suppose we only have "`facility_2″` and `″facility_13″`, and all of three collections. We want to match this waste center with 2 recycling facilities in different regions. Let our criteria be "`cost″`, "`distance″` and "`energy generation capacity″` of the facility. In the Table 3, the approximation degrees of the query `"? -find (facility (2-13), City, collection_1-2-3), normal, medial, good).″` are given.

Table 2: Approximation degrees according to "cheap, close, excellent" criteria.

	Waste Collection Center		
	collection_1	**collection_2**	**collection_3**
Recycling Facility	Facility_11 ,city_F	Facility_11 ,city_F	Facility_11 ,city_F
Approxima tion Degree	0,277	0,25	0,171

Table 3: Approximation degrees according to "normal, medial, good" criteria.

	Waste Collection Center		
	collection_1	**collection_2**	**collection_3**
facility _2	1,0	0,3	0,436
facility_ 13	0,405	0,4	0,405

In the Table 4, the approximation degrees of the query `"? -find (facility_ (2-13), City, collection_ (1-2-3), normal, far, good).″`

Table 4: Approximation degrees according to "normal, far, good" criteria.

	Waste Collection Center		
	collection_1	**collection_2**	**collection_3**
facility_ 2	0,357	1,0	0,245
facility_ 13	0,891	0,17	0,891

As shown in the tables, we fixed three fuzzy criteria and made facility and collection weighting. On the other hand, we did the same weighting by changing the distance criterion. As a result, we found that fuzzy criterion change affects at weighting, but in terms of other criteria, the change was made in the same way.

5 CONCLUSION AND FUTURE WORK

In this study, waste collection is related to artificial intelligence and a sample code is written with fuzzy logical programming language Bousi∼Prolog that is an extension of the standard Prolog language with a fuzzy unification algorithm based on proximity relations. This is to remark that, it is a useful tool for dealing with approximate reasoning and modelling vagueness, also selecting centers with flexible query answering in deductive databases for decision-making process.

As a matter of future work, we should incorporate: graphical tools for helping the programmer to define fuzzy sets; other fuzzy matching options and new application areas such as project management, decision making on environmental-technical criteria.

ACKNOWLEDGEMENTS

This research has been financially supported by Galatasaray University Research Fund (19.402.002).

REFERENCES

Alsinet T., Godo L., 1998. Fuzzy Unification Degree. In *Proc. of the 2nd Intl. Workshop on Logic Programming and Soft Computing'98*, Manchester (UK) , pp 18.

Cayrol M., Farency H., Prade H., 1982. Fuzzy pattern matching. Kybernetes, 11:103-106.

Dubois D., Prade H., Testemale C., 1988. Weighted fuzzy pattern matching. *Fuzzy Sets and Systems*, 28:313-331.

Fontana F.A., Formato F., 1999. Likelog: A logic programming language for flexible data retrieval. In *Proc. of the ACM SAC*, pp. 260–267.

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

Formato F., Gerla G., Sessa M.I., 2000. Similarity-based unification. *Fundam. Inform.* , 41(4):393–414.

Julian P., Rubio C., 2006. A wam implementation for flexible query answering. In A.P.del Pobil, editor, In: *Proc. of the 10th IASTED International Conference on Artificial Intelligence and Soft Computing (ASC 2006)*, August 28-30, 2006, Palma de Mallorca, pages 262–267. ACTA Press.

Julian P., Rubio C., 2009. A similarity-based WAM for Bousi∼Prolog. In: *LNCS*, vol 5517, pp. 245–252. Springer, Heidelberg.

Julian P., Rubio C., 2009. UNICORN: A Programming Environment for Bousi∼Prolog. In: *Proc. of PROLE'09.*

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

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

Julian P., Rubio C., 2010. Bousi~Prolog - A Fuzzy Logic Programming Language for Modeling Vague Knowledge and Approximate Reasoning. In: *ICFC-ICNC*, vol 5517, Valencia, Spain, October 24-26.

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

Mukaidono M., Shen Z.L., Ding L., 1989. Fundamentals of fuzzy Prolog. *Intl. J Approximate Reasons*, 3, pp. 1080–1086.

Orchard R.A., 1998. Fuzzy Clips Version 6.04A. *User's Guide Integrated Reasoning. Institute for Information Technology*. Canada.

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

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

Zadeh L. A., 1965. Fuzzy Sets. Information and Control, 8(3):338–353.

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

APPENDIX

```
%% DIRECTIVES

%% Linguistic variable: cost

   :-domain(cost,0,900,dolar).

:-fuzzy_set(cost,[cheap(200,200,450,600),
normal(300,450,650,700),
expensive(550,750,800,900)]).

%% Linguistic variable: distance

   :-domain(distance,0,120,minutes).

   :-fuzzy_set(distance,[close(0,0,30,80),
medial(20,45,70,90), far(50,85,110,120)]).

%% Linguistic variable: energy generation
capacity

   :-domain(capacity,0,10,capacity).

   :-fuzzy_set(capacity,[fair(0,0,1,3),
good(2,5,7),excellent(6,8,9,10)]).

%% FACTS

%% Facility table

%% facility(Facility_name, City, Cost,
Capacity).
```

```
   facility(facility_1, city_A, cost#400,
fair).

   facility(facility_2, city_A, cost#650,
good).

   facility(facility_3, city_B, cost#900,
extremely#excellent).

   facility(facility_4, city_B, cost#700,
very#good).

   facility(facility_5, city_B, cost#850,
somewhat#good).

   facility(facility_6, city_C, cost#200,
very#fair).

   facility(facility_7, city_D, cost#800,
excellent).

   facility(facility_8, city_D, cost#750,
very#good).

   facility(facility_9, city_E, cost#250,
somewhat#fair).

   facility(facility_10, city_F, cost#700,
very#good).

   facility(facility_11, city_F, cost#550,
somewhat#excellent).

   facility(facility_12, city_G, cost#300,
fair).

   facility(facility_13, city_H, cost#500,
more_or_less#good).

   facility(facility_14, city_H, cost#800,
excellent).

%% Cities table

%% city(Name, Zone)

   city(city_A,zone_1).

   city(city_B,zone_1).

   city(city_C,zone_2).

   city(city_D,zone_3).

   city(city_E,zone_4).

   city(city_F,zone_4).

   city(city_G,zone_4).

   city(city_H,zone_5).

%% Distance table

%% distance(Zone, Collection, Distance)

%% to waste collection center no1

   distance(zone_1,collection_1,medial).

   distance(zone_2,collection_1,somewhat#clo
se).

   distance(zone_3,collection_1,far).

   distance(zone_4,collection_1,extremely#cl
ose).

   distance(zone_5,collection_1,more_or_less
#far).

%% to waste collection center no2

   distance(zone_1,collection_2,very#far).

   distance(zone_2,collection_2,somewhat#clo
se).

   distance(zone_3,collection_2,more_or_less
#medial).

   distance(zone_4,collection_2,somewhat#far
).

   distance(zone_5,collection_2,close).

%% to waste collection center no3

   distance(zone_1,collection_3,more_or_less
#close).

   distance(zone_2,collection_3,extremely#fa
r).

   distance(zone_3,collection_3,somewhat#med
ial).

   distance(zone_4,collection_3,far).

   distance(zone_5,collection_3,more_or_less
#far).

%% RULES

%% close_to(Facility, Collection)

   close_to(Facility, Collection, Distance
):-  facility(Facility, City, _, _),
city(City, Facility_Dist),
distance(Facility_Dist, Collection,
Distance).

   find (Facility, City, Collection, Cost
,Distance , Capacity):-facility(Facility,
City, Cost, Capacity),
close_to(Facility,Collection,Distance).
```