[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases

Sabrine Jandoubi[a], Afef Bahri[b], Nadia Yacoubi Ayadi[c], Salem Chakhar[d]& Ashraf Labib[d]

a Higher Business School of Tunis, University of Manouba, Manouba, Tunisia;
b Miracl Laboratory, Higher Institute of Computer Science and Multimedia, University of Sfax, Sfax, Tunisia;
c RIADI Research Laboratory, National School of Computer Sciences, University of Manouba, Manouba, Tunisia;
d Portsmouth Business School and Centre for Operational Research & Logistics, University of Portsmouth, Portsmouth, UK

ABSTRACT

The rapid development of social networks, intelligent sensors, mobile solutions and Internet of things has led to the emergence of large data sets. Efficiently and effectively exploring these data sets is a challenging question, especially when the imperfectness of real-world needs to be taken into account. The objective of this paper is thus to propose several solutions for modelling, representing and implementing imperfect information within large fuzzy databases. More specifically, in this paper imperfect information is modelled through a series of generic fuzzy data types, uniformly represented by means of possibility distributions and implemented using the basic constructs of object databases. These solutions are particularly useful for exploring large fuzzy databases since they permit to minimize the space required to store imperfect information, and access efficiently and effectively these databases.

Keywords: large data sets; imperfect information; uncertainty; imprecision; fuzziness; incomplete data; fuzzy attributes; fuzzy database; mapping rule

Introduction

The digital era is characterized by large data sets resulting from the rapid and worldwide development of social networks, intelligent sensors, mobile solutions and Internet of things. These heterogeneous data sets are stored in organizations, under various forms as structured data like data warehouse and unstructured data. Filtrating,

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

organizing and extracting pertinent information from these data sets in order to help different users in many areas use efficiently and effectively is a challenging question, especially when the imperfectness ─ including uncertainly, imprecision and fuzziness ─ of real-world needs to be taken into account.

In fuzzy databases, attributes may be crisp or fuzzy. The crisp attributes are defined and implemented as with the conventional databases. The fuzzy attributes are domain-specific and most of them are not supported by conventional databases. Although there are a large number of fuzzy data models and databases that have been proposed in the literature (see, e.g., Berzal et al., 2007; Bosc et al, 2005; Cadenas et al., 2010, 2014; Cuevas et al., 2008; Singh et al., 2014; Ma and Yan, 2010; Ma et al., 2014; Yan and Ma, 2014a, 2014b), these models fail to respond adequately to the characteristics of the digital age: very large data sets that evolve rapidly and an increased need for the organizations to extract crucial knowledge and data pertinent for decision making from these data sets. Having timely the required and relevant information and knowledge is considered as a critical issue for the organizations.

Thus, the objective of this paper is to propose several solutions to handle the uncertainly, imprecision and fuzziness of knowledge and data within large and evolving data sets. More specifically, we propose several solutions for modelling, representing and implementing imperfect information within large fuzzy databases. In this paper, imperfect information is modelled through a series of generic fuzzy data types, uniformly represented by means of possibility distributions and implemented using the basic constructs of object databases. For implementation purposes, we designed a set of rules in order to map the different fuzzy attributes from the initial fuzzy data model into the destination fuzzy database model. These rules are generic permitting to implement

the fuzzy data types using any database system that supports the basic constructs of object databases (see, e.g., Singh et al. (2014)), especially multi-valued attributes, composed attributes and array data types.

These solutions are particularly useful for exploring large fuzzy databases since they minimize the space required to store imperfect information, and permit to access efficiently and effectively these databases. Indeed, these solutions offer: (i) an uniformed way to model and represent a large set of fuzzy data types through possibility distributions; (ii) a reduced number of meta-relations and relations to store the fuzzy data and their characteristics; and (iii) a reduced access and exploration time for extracting pertinent information from these data sets.

The rest of the paper goes as follows. The next section deals with imperfect information modelling and representation. The third section provides the mapping rules. The fourth section focuses on the implementation aspects. The fifth section presents the performance analysis and a comparative study. The sixth section discusses some implementation issues. The last section concludes the paper.

Modelling and representation of imperfect information

Different terms have been used in the fuzzy database literature to design information which is not crisp and there are several attempts to classify them (e.g., Cadenas et al, 2011; Klir and Yuan, 2005; Ma, 2005; Rodrigues et al, 2009). The term ‘imperfection’ has been introduced by Motro (1990) to indicate imprecision, vagueness, uncertainty and inconsistency. In Bosc and Prade (1990), the authors distinguish five types of imperfect information: inconsistency, imprecision, vagueness, uncertainty and ambiguity. In this paper, the term ‘imperfect information’ is used to design all kinds of

non crisp information, especially imprecision, vagueness and uncertainty.

In the rest of this section, we provide a rich set of fuzzy data types permitting to model almost all kinds of imperfect information. This list has been established based on the works of Bahri et al (2005), Medina et al (1995) and Rodrigues et al (2009). The fuzzy data types are organized according to their domains into three groups: (i) fuzzy data types defined over ordered domains, (ii) fuzzy data types defined as linguistic labels, and (iii) fuzzy data types defined over unordered domains. We have also added a fourth group relative to (iv) incomplete data, which includes three specific values, namely unknown, undefined or null, that may be taken by fuzzy data types. The different fuzzy data types and values introduced in what follows are uniformly represented by means of possibility distributions.

Fuzzy data types defined over ordered domains

This group contains fuzzy data types having possibility distribution defined on an ordered discrete or continuous domain. Each data type of this group is associated with a *degree of membership* (d.o.m) function. A brief description of these data types follows.

Fuzzy Range. The Fuzzy Range data type is represented through as a trapezoidal possibility distribution. This data type handles the 'more or less' information between two numeric values. The graphical representation of possibility distribution of this data type is given by Model I.1 in Figure 1 and may be written as $\{\mu(z)/z : z \in D\}$ where D is the domain of the data type values and $\mu(z)$ is the d.o.m of z in the fuzzy set on which the data type is defined. As it is shown in this figure, four parameters are required to define the possibility distribution of this data type: α, β, γ and λ. The parameters α and β represent the support of the fuzzy set associated with the data type values and γ and

λ represent the limits of the transition zones.

Approximate Value. The Approximate Value data type handles the imprecise concept 'approximately *n*' where *n* is a number. The triangular graphical representation of possibility distribution of this data type is illustrated by Model I.2 in Figure 1 and may be written as $\{\mu(z)/z:z\in D\}$. Here, three parameters are required: the central value of the concept *c*, limit of left transition zone c^- and the limit of right transition zone c^+.

Interval. The Interval data type is a special case of Fuzzy Range data type that represents the classical crisp range. It is represented graphically by Model I.3 in Figure 1. Mathematically, the possibility distribution of Interval data type is written as $\{\mu(z)/z:z\in D\}$. The parameters required here are the limits of the range α and β.

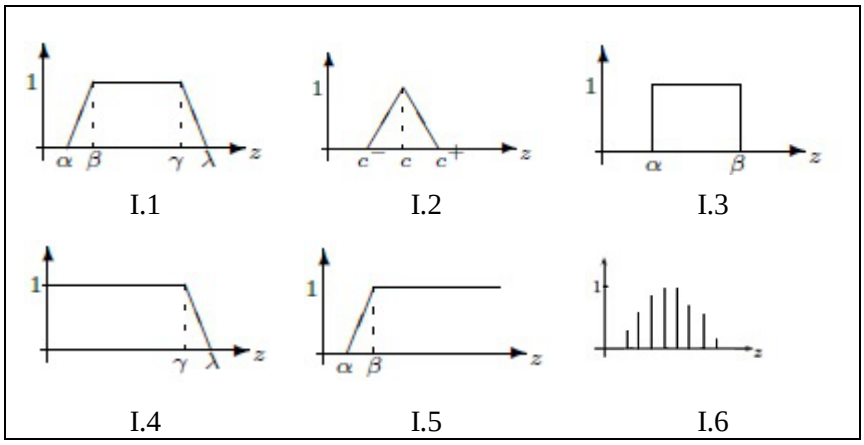


Figure 1 Fuzzy data types defined over ordered domains

Less Than/More Than. The 'Less Than' and 'More Than' data types focalize only on one side of a value. The graphical representations of the possibility distributions of 'Less Than' and 'More Than' data types are shown in Models I.4 and I.5 in Figure 1, respectively. Mathematically, the possibility distribution associated with both of them may be written as $\{\mu(z)/z:z\in D\}$. Two parameters are required to define these data

types: the value of interest (γ or β) and the limit of the transition range (λ or α).

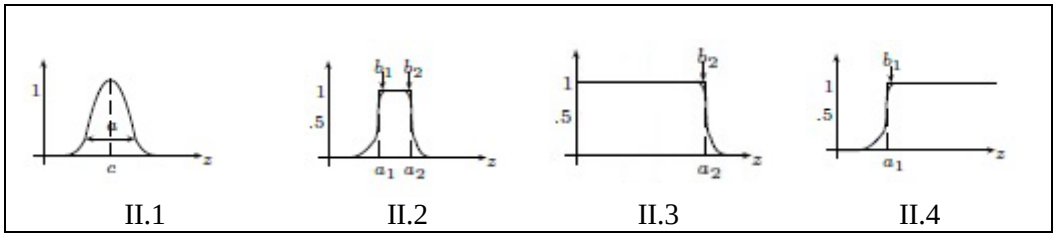
Discrete Possibility Distribution. A Discrete Possibility Distribution (or simply Possibility Distribution) is defined over an ordered set of numeric values. This data type is often represented as $\{p_1/d_1,...,p_n/d_n\}$ with $p_i \leq p_{i+1}$, d_i is a numerical value and $p_i \in [0,1]$ is the possibility that the value of the concept under consideration is equal to d_i. Generally the p_i must sum to 1 to obtain a normalized distribution. The characteristics of this data type are given by Model I.6 in Figure 1. The graphical representation of a Possibility Distribution takes the form of a histogram where the height of the *i*th element of the histogram (relative to the value d_i of the concept considered) is the possibility value p_i.

Fuzzy data types defined as linguistic labels

Some imperfections are expressed in terms of linguistic labels that refer to imprecise concepts which may be associated with a possibility distribution. Different models of possibility distributions can be used to represent linguistic labels. Figure 2 presents graphical representation of four common linguistic labels. Model II.1 in Figure 2 represents the Gaussian model. The parameters required here are: the central value of the data type c and the parameter a that governs the shape of the d.o.m. Model II.2 in Figure 2 is an extension of the previous one that applies when the central value of the concept may take a range of values instead of only one value. Four parameters are required here: the limits of the central range a_1 and a_2; and the left and right transition zones b_1 and b_2, respectively. Note that a_1 and a_2 are the crossover (or transition) points defined such that $\mu(a_1)=\mu(a_2)=0.5$. Models II.3 and II.4 in Figure 2 are the asymmetric extensions of Model II.1 that apply when only the left or right side of the concept is of interest. The required parameters are a_2 and b_2 for Model II.3; and a_1 and b_1 for Model

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

II.4.



II.1 II.2 II.3 II.4

Figure 2 Fuzzy data types defined as linguistic labels

The mathematical representation of all these data types is $\{\mu(z)/z:z\in D\}$. Attributes defined as linguistic labels need also to be associated with proximity relations defined on their domains. Each linguistic label may be represented by its membership function. If the reference set is a finite set, the d.o.m are discrete values defined in [0,1]. If the reference set is infinite, we can represent the d.o.m values as continuous membership functions.

Fuzzy data types defined on unordered domains

This group contains fuzzy data types constructed on unordered discrete domains. By unordered domain we mean that the order of the possible values has no importance. Some of the data types of this group require the definition of similarly or resemblance relationships on their domains.

Single Scalar. The Single Scalar data type corresponds to simple linguistic values with no explicit possibility distribution. However, a possibility distribution of a simple scalar *s* may be defined as {1/*s*}. A proximity relation is often associated with this data type. The Single Scalar data type is designed by Model III.1 in the rest of the paper.

Simple Number. The Simple Number is a crisp data type which is handled as in

conventional databases. The possibility representation of a Simple Number *n* is {1.0/*n*}. This Simple Number data type is designed by Model III.2 in the rest of the paper.

Possible Scalar Assignments. The Possible Scalar Assignments data type permits to handle data types defined on a set of scalars. For example, the height of a person may be defined as follows: *Height*= {*tall*, *very tall*}. We can associate to this data type the possibility distribution {1/*tall*, 1/*very tall*}. A proximity relation is often defined on the domain of this data type. The Possible Scalar Assignments data type is designed by Model III.3 in the rest of the paper.

Possible Numeric Assignments. The Possible Numeric Assignments is similar to the previous one. It differs only on the fact that it is defined on a set of numeric values. For example, the height of a person may be defined as follows: *Height*= {1.85, 1.95}. It can be represented through the possibility distribution {1/1.85, 1/1.95}. This type is designed by Model III.4 in the rest of the paper.

Incomplete data

To model incompleteness in fuzzy databases, some data types may take some specific values, namely Unknown, Undefined or Null.

Unknown. This data value means that we cannot decide which is the value of the data type among several plausible values. But the data type may take any value from its domain. The possibility distribution of the Unknown data value is {1/*u*:*u*∈*D*}. In the rest of this paper, an unknown data value is denoted by *unk* and called Model IV.1. The graphical representation of an unknown value is given in Figure 3.

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

Undefined. This data value means that there is not any defined value that can be assigned to the data type. This means that no one of the domain values is authorized. The possibility distribution of undefined data value is $\{0/u:u \in D\}$. In the rest of this paper, an undefined value is denoted by *und* and called Model IV.2. The graphical representation of an undefined value is given in Figure 3.

Null. This data value means that we cannot even know whether the value is unknown or undefined. The possibility distribution of this data type is $\{1/unk, 1/und\}$. In the rest of this paper, an undefined data value is denoted by *nil* and called Model IV.3.

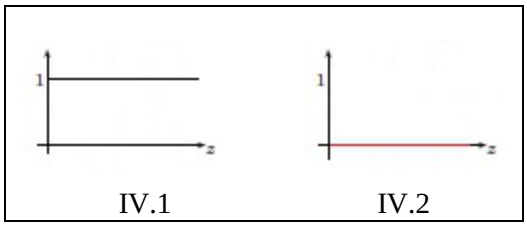


Figure 3 Graphical representation of unknown and undefined values

Specification and transformation of attributes

In that follows, we denote by *M* the original fuzzy data model and by *T* the destination fuzzy database model. An attribute in *M* is basically characterized by its name, data type and domain. There are other system attributes but only these ones are considered in this paper. The domain of an attribute is the set of values the attribute may take. Let *D*(*Attr*) denotes the domain of attribute *Attr*. An attribute may be crisp or fuzzy. A crisp attribute may take any conventional data type (e.g., integer, string). The data type of a fuzzy attribute may be any one from the list introduced earlier. In addition to its basic data type, a fuzzy attribute is characterized by a set of parameters permitting to generate its possibility distribution. In the most general case, the values of a fuzzy attribute *Attr*

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

belong to $D(Attr) \cup \{unk, und, nil\}$.

The mapping of an attribute $Attr \in M$ into T concerns both the extensional level and intensional level of the destination fuzzy database T. The first one is related to the transformation of the attribute during the creation of the destination fuzzy database T. The second is relative to the characteristics of the attributes.

Extensional level

At the extensional level, the crisp attributes are mapped as in conventional database models. In turn, the fuzzy attributes are mapped into different types of attributes, along with their fuzzy data types. In what follows, we provide a series of rules for mapping crisp and fuzzy attributes at the extensional level.

The following mapping rule applies to attributes defined as crisp data types.

> Mapping Rule 1. Let *Attr* be a crisp attribute in *M*. Attribute *Attr* is mapped in *T* as a crisp attribute with the same name and characteristics.

Fuzzy attributes defined according to Models I.1 to I.5 or Models II.1 to II.4 are mapped according to the following rule.

> Mapping Rule 2. Let *Attr* be a fuzzy attribute in *M* defined through Models I.1 to I.5 or Models II.1 to II.4. The attribute *Attr* is mapped in *T* into a new composed attribute with the same name and composed of the following elements: (i) *Value*: which is devoted to store the value of the attribute as provided by the user; (ii) *DataType*: which is the data type of the attribute provided by the user; and (iii) *ParametersList*: which is a multi-valued attribute indicating the list of parameters' values needed to generate the possibility distribution of the value specified in the first component.

According to this definition, at the extensional level, a fuzzy attribute defined according to Models I.1 to I.5 or Models II.1 to II.4 can accept crisp as well as fuzzy

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

values. Hence, the component *DataType* can be either crisp or fuzzy. However, we note that when the value of the attribute is crisp, the component *ParametersList* will be empty.

Fuzzy attributes defined as Discrete Possibility Distribution data type (Model I.6) are mapped into a two-dimensional list of the form $((p_1,d_1),...,(p_n,d_n))$ with $p_i \leq p_{i+1}$, d_i is a numerical value and $p_i \in [0,1]$ is the possibility that the value of the concept under consideration is equal to d_i, e.g., *Age*=((0.5,20),(1,21),(0.7,22),(0.3,23)). The mapping of fuzzy attributes defined according to Model I.6 is formalized by the following rule.

> Mapping Rule 3. Let *Attr* be a fuzzy attribute in *M* defined through Model I.6. Then, the attribute *Attr* is mapped in *T* into an attribute with the same name and accepting values defined as a two-dimensional list of the form $((p_1,d_1)),...,((p_n,d_n))$ where p_i are d_i are as defined above.

Fuzzy attributes defined as Single Scalars (e.g., *Quality=average*) (see Model III.1) with pre-defined domains are mapped according to the following rule.

> Mapping Rule 4. Let *Attr* be a fuzzy attribute in *M* defined through Model III.1. Then, the attribute *Attr* is mapped in *T* into an attribute with the same name and of text data type with a pre-defined domain.

Fuzzy attributes defined as a Possible Scalar Assignments Set (e.g., *Quality*={*bad*, *average*, *good*}) or a Possible Numerical Assignments Set (e.g., *Age*={20,21,22,23}) are mapped according to the following rule.

> Mapping Rule 5. Let *Attr* be a fuzzy attribute in *M* defined through Models III.3 or III.4. Then, the attribute *Attr* is mapped in *T* into an attribute with the same name and defined as one-dimensional list of scalar (for Model III.3) or numerical (for Model III.4) values.

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

Intensional level

The characteristics of the attributes need to be added to the intensional level of the destination fuzzy database *T*. This requires the creation of several metadata relations. In the rest of this section, we first detail the mapping of the basic characteristics of crisp as well as fuzzy attributes. Then, we present the mapping of the characteristics of the linguistic labels and the proximity relations that can be associated with some fuzzy attributes.

Mapping of the basic characteristics of attributes

To maintain the basic characteristics of both crisp and fuzzy attributes, we propose to define three meta-relations: ATTRIBUTES, SYM-ATTRIBUTES and FUZZ-ATTRIBUTES. The meta-relation ATTRIBUTES is devoted to store the characteristics of crisp attributes. At least the flowing attributes should be maintained: (i) *AttrID*: it uniquely identifies each attribute; (ii) *AttrName*: the name of the attribute; (iii) *RelationID*: denotes the fuzzy relation to which the attribute belongs; (iv) *DataType*: the crisp data type of the attribute (it may take the values of integer, real, float, etc.); (v) *IsMultiValued*: indicates if the attribute is multi-valued or not; and (vi) *IsKey*: says if the attribute is a key or part of a key or not.

> Mapping Rule 6. Let *Attr* be a crisp attribute in *M*. The characteristics of attribute *Attr* are added to meta-relation ATTRIBUTES in *T* with the same structure given above.

The meta-relation SYM-ATTRIBUTES is devoted to store the characteristics of symbolic attributes. It contains the same attributes as the meta-relation ATTRIBUTES (i.e., *AttrID*, *AttrName*, *RelationID*, *DataType*, *IsMultiValued* and *IsKey*) with the

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

same definition as above and adds a new multi-valued attribute called *Domain* devoted to specify the possible values of the attribute. These values should have the same basic data type indicated in *DataType* attribute.

> Mapping Rule 7. Let *Attr* be a symbolic attribute in *M*. The characteristics of attribute *Attr* are added to meta-relation SYM-ATTRIBUTES in *T* with the same structure given above.

The meta-relation FUZZ-ATTRIBUTES is devoted to store the characteristics of fuzzy attributes. It is structured as follows: (i) *AttrID*: it uniquely identifies each attribute; (ii) *AttrName*: the name of the attribute; (iii) *CompAttrName*: a component attribute of *AttrName*, which, as specified by Mapping Rule 2, can take one of the following three values: *Value*, *DataType* and *ParametersList*; (iv) *RelationID*: the fuzzy relation to which the attribute belongs; (v) *DataType*: the data type of the component attribute; (vi) *IsMultiValued*: indicates if the attribute is multi-valued or not; and (vii) *FuzzyDomain*: the domain of fuzzy attribute. The attribute *DataType* may take a crisp data type (i.e., defined as integer, real, float, etc.) or a fuzzy data type from the list given in the second section. In the first case, the attribute *DataType* and *FuzzyDomain* will have the same value.

> Mapping Rule 8. Let *Attr* be a fuzzy attribute in *M*. The characteristics of attribute *Attr* are added to meta-relation FUZZ-ATTRIBUTES in *T* with the same structure given above.

Mapping of the characteristics of linguistic labels

The domains of some fuzzy data types may require the specification of a set of linguistic labels. To maintain the characteristics of these linguistic labels, we propose to define a meta-relation called LABELS with the following attributes: (i) *AttrID*: it

uniquely identifies each attribute; (ii) *LabelID*: the label identifier, which is unique for each attribute; (iii) *Label*: the label value; (iv) *Model*: indicates the model used to define the membership function of the linguistic label; and (v) *ParametersList*: which is a multi-valued attribute used to maintain the list of parameters needed to generate the possibility distributions of linguistic labels.

> Mapping Rule 9. Let *Attr* be a fuzzy attribute in *M*. If the domain of *Attr* is defined as a set of linguistic labels, then add the characteristics of these linguistic labels to meta-relation LABELS in *T* with the same structure given above.

It is important to note that the parameters specified in *ParametersList* of the meta-relation LABELS depend on the application domain and should be specified by the domain's expert.

Mapping of the proximity relations

Some fuzzy data types may require the definition of a proximity relation on their domains. To maintain the characteristics of these proximity relations, we propose to define a meta-relation PROXIMITY with the following attributes: (i) *AttrID*: references the attribute for which the proximity relation is defined; (ii) *Label1ID* and (iii) *Label2ID*: denote two linguistic terms belonging to the attribute's domain; and (iv) *Degree*: stores the similarity degree between *Label1ID* and *Label2ID*.

> Mapping Rule 10. Let *Attr* be a fuzzy attribute in *M* with fuzzy domain defined as a set of linguistic labels. If the linguistic labels are associated with a proximity relation, then the characteristics of these proximity relations should be added to meta-relation PROXIMITY in *T* with the same structure given above.

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

Implementation

The proposed mapping rules have been implemented on the Object-Relational Database Management System PostgreSQL.

Database example

For illustration purposes, we will use the database example given in Figure 4. This example is a simple extract reproduced from the fuzzy data model given in Jandoubi et al (2015a). Although this example is initially based on the Fuzzy Semantic Model (Bouaziz et al, 2007), the solutions proposed in this paper are generic and can apply to any fuzzy data model. We note in particular that in Jandoubi et al (2015a), GALAXY, STAR, SUPERNOVA and PERSON are defined as classes. This is not a requirement in the sense that our solutions may apply to fuzzy classes or fuzzy relations with no modification.

GALAXY(GalaxyName, **Age**, **Location**);
STAR(StarName, *TypeStar*, **Age**, **Location**, **Luminosity**, **Weight**);
SUPERNOVA(SNName, *TypeSNova*, **Luminosity**, **Weight**);
PERSON(Name, **Age**, ResearchField);

Figure 4 Database example

In Figure 4, fuzzy attributes are bold-faced and symbolic attributes are italic-faced. The remaining attributes are crisp. The underlined attributes are the keys. For example, STAR has one crisp attribute (*StarName*), one symbolic attribute (*TypeStar*) and four fuzzy attributes (*Age*, *Location*, *Luminosity* and *Weight*).

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

The database level

Fuzzy data types

In this research project, the different data types given in the second section that require at least one parameter are implemented as Fuzzy Abstract Data Types (FADT). In such a way, these FADTs can be used for instantiating different fuzzy data types associated with different relations/classes in the same application domain or even in different application domains. For instance, a FADT implementing an Approximate Value can be used to define the attribute *Age* associated with STAR in Figure 4 and another attribute called *Age* associated with PERSON. We need simply to use different parameters' values to instantiate the FADT implementing the Approximate Value.

The new data types in PostgreSQL are created using the CREATE TYPE command. There are three different forms for this command. The first form creates a composite type. The composite type is specified by a list of attribute names and data types. The second form creates an enumerated type. Enumerated types take a list of one or more quoted labels. The third form creates a new base type. The first form is used here to define fuzzy data types while the second form is used to define domains of linguistics labels and symbolic attributes. The third, more advanced, form of CREATE TYPE is very useful in creating fuzzy data types. However, in this paper, fuzzy data types are defined as composite data types using the first form of CREATE TYPE. Nevertheless, we intend use the advanced form in our future work.

The generic syntax of the first form of CREATE TYPE command is as follows:

CREATE TYPE *name* AS (*AttributeName datatype* [, ...])

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

Hence, composite types are simply defined as collection of attribute names and data types. This is essentially the same as the row type of a table, but using CREATE TYPE avoids the need to create an actual table when all that is wanted is to define a type. We provide in Listing 1 some illustrative examples for creating FADT related to the mapping of the data model of Figure 4. The two first examples permit to create the Fuzzy Range data type. As indicated earlier, the possibility distribution of Fuzzy Range data types are defined through four parameters that we denoted by α, β, γ and λ. The basic domain on which these parameters are defined may be integer or real. For this purpose, we defined two types of Fuzzy Range in Listing 1: *IntegerFuzzyRange* and *RealFuzzyRange*. The first type is integer-based Fuzzy Range data type where all the parameters should be integer. The second type is real-based Fuzzy Range data type where all parameters are defined as real. This distinction between integer and real domains is maintained also for the definition of Approximate Value and Interval data types (see Listing 1). The same description given above still applies here but with different number of parameters (three for Approximate Value data type and two for Interval data type).

```
CREATE TYPE IntegerFuzzyRange AS (Alpha integer, Beta integer, Gamma
integer, Delta integer);
CREATE TYPE RealFuzzyRange AS (Alpha real, Beta real, Gamma real, Delta
real);

CREATE TYPE IntegerApproximateValue as (MaringLeft integer, CentralValue
inetger, MarginRight integer);
CREATE TYPE RealApproximateValue AS (MarginLeft real, CentralValue real,
MarginRight real);

CREATE TYPE IntegerInterval AS (Alpha integer, Beta integer);
CREATE TYPE RealInterval AS (Alpha real, Beta real);
```

Listing 1 Creating fuzzy ADTs

The use of these fuzzy data types for creating fuzzy attributes is discussed and illustrated later.

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

Fuzzy and symbolic domains

The domain of linguistic labels and symbolic attributes are defined and implemented using the second form of CREATE TYPE command. The generic syntax of the second form of this command is as follows:

CREATE TYPE *name* AS ENUM (['*label*' [, ...]])

Accordingly, enumerated types are simply defined as a list of one or more quoted labels. We provide in Listing 2 and Listing 3 some illustrative examples for creating fuzzy and symbolic domains related to the mapping of the data model given in Figure 4. The two first examples in Listing 2 permit to create fuzzy age domains. Although these two domains have the same list of linguistic labels, they are different since these linguistic labels have different parameters' values. The two next enumerated types define two domains associated with Luminosity and Location fuzzy attributes.

```
CREATE TYPE StarAgeDomain AS ENUM('very young', 'young', 'old', 'very
old');
CREATE TYPE PersonAgeDomain AS ENUM ( 'very young', 'young', 'old', 'very
old');
CREATE TYPE LuminosityDomain AS ENUM ('very low', 'low', 'medium', 'high',
'very high');
CREATE TYPE LocationDomain AS ENUM ('near', 'very near', 'distant', 'very
distant');
```

Listing 2 Creating fuzzy domains

Listing 3 shows the definition and implementation of two symbolic domains using the second form of CREATE TYPE command. These symbolic domains are associated with attributes *TypeStar* and *TypeSNova*, respectively.

```
CREATE TYPE TypeStarDomain AS ENUM ('nova', 'supernova');
CREATE TYPE TypeSNovaDomain AS ENUM ('Ia', 'Ib', 'Ic', 'Ib/c', 'Ic/b',
'II-P', 'II-L');
```

Listing 3 Creating symbolic domains

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

The use of these fuzzy and symbolic domains for creating fuzzy attributes is discussed and illustrated later.

Fuzzy attributes

Implementation of Models I.1 to I.5 and II.1 to II.4. As established by Mapping Rule 2, fuzzy attributes defined through Models I.1 to I.5 or II.1 to II.4 are mapped into new composite attributes with the following components: (i) *Value*: which is the value of the attribute as provided by the user; (ii) *DataType*: which is the fuzzy data type of the attribute provided by the user; and (iii) *ParametersList*: which is a multi-valued attribute indicating the list of parameters' values needed to generate the possibility distribution of the fuzzy data type.

Let *Attr* be a fuzzy attribute defined through Models I.1 to I.5 or II.1 to II.4. Then, the mapped attribute from *Attr* is created as follows:

CREATE TYPE *Attr* AS (
Value fuzzy-data-type|*linguistic-label*,
DataType basic-data-type,
ParametersList basic-data-type[]|*fuzzy-data-type-param*);

The symbol | above sets for OR. The *Value* component can be specified as a fuzzy data type (*fuzzy-data-type*) or as a fuzzy domain that is defined through a series of linguistic labels (*linguistic-label*). The first case applies mainly for non-linguistic label-based attributes (e.g., the attribute *Age* associated with GALAXY which is defined as an Approximate Value). The second case is more useful for linguistic label-based attributes where a set of possible linguistic labels is explicitly defined by the user (e.g., the

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

attribute *Location* associated with GALAXY which is defined as a Linguistic Label). The *DataType* component takes the basic data type (*basic-data-type*) provided by the user. The value of this attribute may be any conventional (crisp) data type. The *ParametersList* component can be defined either as list of the basic data type (*basic-data-type*) or a composite attribute with a predefined number of parameters (*fuzzy-data-type-param*). In the first case, the number of parameters is not specified. In the second case, the number of parameters is implicitly defined by the number of the components in the composite data type *fuzzy-data-type-param*.

Listing 4 provides some illustrative examples for creating fuzzy attributes using the syntax given above.

```
CREATE TYPE StarAge AS (Value StarAgeDomain, DataType real, ParametersList
real[]);

CREATE TYPE PersonAge AS (Value PersonAgeDomain, DataType integer,
ParametersList IntegerApproximateValue);

CREATE TYPE Luminosity AS (Value LuminosityDomain, DataType real,
ParametersList real[]);

CREATE TYPE Location AS (Value LocationDomain, DataType real,
ParametersList RealFuzzyRange);
```

Listing 4 Creating fuzzy attributes as composite types

Implementation of Model I.6. Based on Mapping Rule 3, fuzzy attributes defined through Model I.6 are mapped as a list where each element is composed of two numbers. Let *Attr* be a fuzzy attribute defined through Model I.6. Then, the mapped attribute from *Attr* is created as follows:

Attr numerical[][2];

According to this sentence, fuzzy attribute defined through Model I.6 are defined as two-dimensional array data type where each element is a list of two numbers. An array data type is named by appending square brackets ([]) to the data type name of

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

the array elements. The above command will create an attribute named *Attr* as a two-dimensional array of type numerical where each element is of the form (p_i,d_i) where d_i is a numerical value and $p_i \in [0,1]$ is the possibility that the value of the concept under consideration (i.e., represented by attribute *Attr*) is equal to d_i. More specific numerical data types (e.g., integer, real, float, etc.) can also be used.

The first example in Listing 5 shows the definition of a fuzzy attribute called *Age*, which is defined as a two-dimensional list. This attribute accepts values defined as possibility distributions.

```
Age real [][2];
Age PersonAgeDomain;
Height=HeightDomain[];
height= real[];
```

Listing 5 Creating fuzzy attributes as list or as scalar

Implementation of Model III.1. According to Mapping Rule 4, fuzzy attributes defined through Model III.1 is mapped as a text type with a pre-defined domain. Let *Attr* be a fuzzy attribute defined through Model III.1. Then, the mapped attribute from *Attr* is created as follows:

Attr enum-domain;

where *enum-domain* is an enumerated domain defined using CREATE TYPE command as examined in Section 4.2.2. The second example in Listing 5 shows the definition of a fuzzy attribute *Age* defined based on an enumerated domain called *PersonAgeDomain*. The latter, which sets the domain of a person's age, has been defined in Listing 2. This attribute accepts values defined as a linguistic label from the pre-defined domain, e.g., *Age=young*.

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

Implementation of Model III.3. According to Mapping Rule 5, fuzzy attributes defined through Model III.3 are mapped as a list of scalar values. Let *Attr* be a fuzzy attribute defined through Model III.3. Then, the mapped attribute from *Attr* is created as follows:

Attr text[];

According to this definition, fuzzy attributes defined through Model III.3 are defined as one-dimensional array data type where each element is of text data type. More specific textual data types (e.g., varchar, etc.) can also be used. The third example in Listing 5 shows the definition of a fuzzy attribute called *Height* defined as a set of scalar values from a pre-defined list called *HeightDomain* that can be specified as follows:

CREATE TYPE *HeightDomain* AS ENUM ('*very short*','*short*', '*tall*','*very tall*');

The attribute *Height* accepts a set of values from *HeightDomain*, e.g., *Height*={*tall*, *very tall*}.

Implementation of Model III.4. According to Mapping Rule 5, fuzzy attributes defined through Model III.4 are mapped as a list of numerical values. Let *Attr* be a fuzzy attribute defined through Model III.4. Then, the mapped attribute from *Attr* is created as follows:

Attr numerical[];

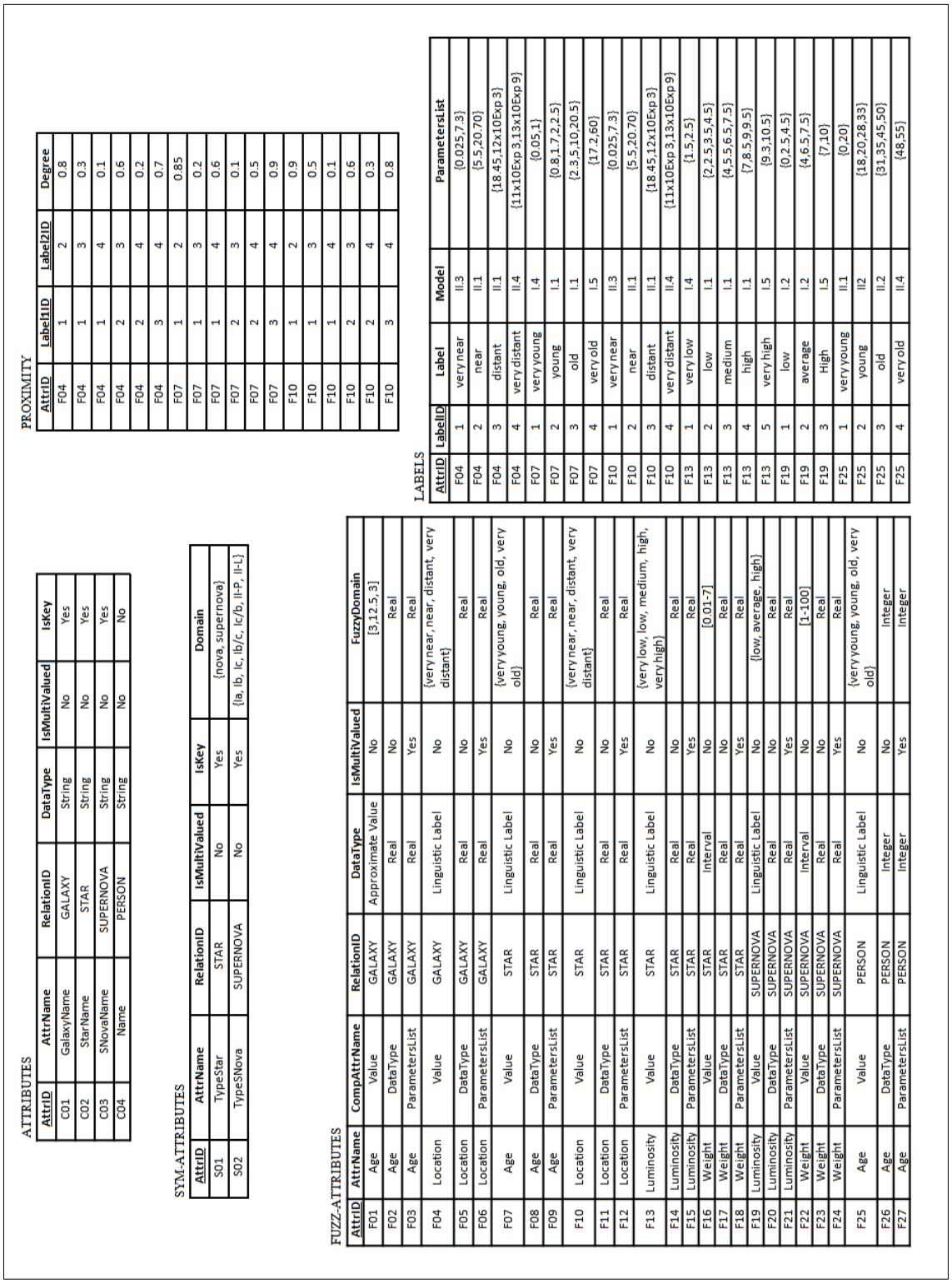
According to this definition, fuzzy attributes defined through Model III.4 are defined as one-dimensional array of numerical data type. More specific numerical data types (integer, real, float, etc.) can also be used. The fourth example in Listing 5 shows the definition of a fuzzy attribute called *Height* defined as a list of real numbers. The attribute Height accepts a set of real values, e.g. *Height*={1.85,1.95}.

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

The metadata level

The characteristics of attributes need to be stored in the metadata level. As discussed in the third section, five meta-relations are required: ATTRIBUTES, SYM-ATTRIBUTES, FUZZ-ATTRIBUTES, LABELS and PROXIMITY. The extensional definitions of the meta-relations associated with Figure 4 are given in Figure 5.

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232



ATTRIBUTES

AttriD	AttrName	RelationID	DataType	IsMultiValued	IsKey
C01	GalaxyName	GALAXY	String	No	Yes
C02	StarName	STAR	String	No	Yes
C03	SNovaName	SUPERNOVA	String	No	Yes
C04	Name	PERSON	String	No	No

SYM-ATTRIBUTES

AttriD	AttrName	RelationID	IsMultiValued	IsKey	Domain
S01	TypeStar	STAR	No	Yes	{nova, supernova}
S02	TypeSNova	SUPERNOVA	No	Yes	{Ia, Ib, Ic, Ib/c, Ic/b, II-P, II-L}

FUZZ-ATTRIBUTES

AttriD	AttrName	CompAttrName	RelationID	DataType	IsMultiValued	FuzzyDomain
F01	Age	Value	GALAXY	Approximate Value	No	[3,12.5,3]
F02	Age	DataType	GALAXY	Real	No	Real
F03	Age	ParametersList	GALAXY	Real	Yes	Real
F04	Location	Value	GALAXY	Linguistic Label	No	{very near, near, distant, very distant}
F05	Location	DataType	GALAXY	Real	No	Real
F06	Location	ParametersList	GALAXY	Real	Yes	Real
F07	Age	Value	STAR	Linguistic Label	No	{very young, young, old, very old}
F08	Age	DataType	STAR	Real	No	Real
F09	Age	ParametersList	STAR	Real	Yes	Real
F10	Location	Value	STAR	Linguistic Label	No	{very near, near, distant, very distant}
F11	Location	DataType	STAR	Real	No	Real
F12	Location	ParametersList	STAR	Real	Yes	Real
F13	Luminosity	Value	STAR	Linguistic Label	No	{very low, low, medium, high, very high}
F14	Luminosity	DataType	STAR	Real	No	Real
F15	Luminosity	ParametersList	STAR	Real	Yes	Real
F16	Weight	Value	STAR	Interval	No	[0.01-7]
F17	Weight	DataType	STAR	Real	No	Real
F18	Weight	ParametersList	STAR	Real	Yes	Real
F19	Luminosity	Value	SUPERNOVA	Linguistic Label	No	{low, average, high}
F20	Luminosity	DataType	SUPERNOVA	Real	No	Real
F21	Luminosity	ParametersList	SUPERNOVA	Real	Yes	Real
F22	Weight	Value	SUPERNOVA	Interval	No	[1-100]
F23	Weight	DataType	SUPERNOVA	Real	No	Real
F24	Weight	ParametersList	SUPERNOVA	Real	Yes	Real
F25	Age	Value	PERSON	Linguistic Label	No	{very young, young, old, very old}
F26	Age	DataType	PERSON	Integer	No	Integer
F27	Age	ParametersList	PERSON	Integer	Yes	Integer

PROXIMITY

AttriD	Label1ID	Label2ID	Degree
F04	1	2	0.8
F04	1	3	0.3
F04	1	4	0.1
F04	2	3	0.6
F04	2	4	0.2
F04	3	4	0.7
F07	1	2	0.85
F07	1	3	0.2
F07	1	4	0.6
F07	2	3	0.1
F07	2	4	0.5
F07	3	4	0.9
F10	1	2	0.9
F10	1	3	0.5
F10	1	4	0.1
F10	2	3	0.6
F10	2	4	0.3
F10	3	4	0.8

LABELS

AttriD	LabelID	Label	Model	ParametersList
F04	1	very near	II.3	{0.025,7.3}
F04	2	near	II.1	{5.5,20.70}
F04	3	distant	II.1	{18.45,12x10Exp3}
F04	4	very distant	II.4	{11x10Exp3,13x10Exp9}
F07	1	very young	I.4	{0.05,1}
F07	2	young	I.1	{0.8,1.7,2,2.5}
F07	3	old	I.1	{2.3,5,10,20.5}
F07	4	very old	I.5	{17,2,60}
F10	1	very near	II.3	{0.025,7.3}
F10	2	near	II.1	{5.5,20.70}
F10	3	distant	II.1	{18.45,12x10Exp3}
F10	4	very distant	II.4	{11x10Exp3,13x10Exp9}
F13	1	very low	I.4	{1.5,2.5}
F13	2	low	I.1	{2,2.5,3.5,4.5}
F13	3	medium	I.1	{4.5,5.6,5.7,5}
F13	4	high	I.1	{7,8.5,9,9.5}
F13	5	very high	I.5	{9,3,10.5}
F19	1	low	I.2	{0.2,5,4.5}
F19	2	average	I.2	{4,6.5,7.5}
F19	3	High	I.5	{7,10}
F25	1	very young	II.1	{0,20}
F25	2	young	II2	{18,20,28,33}
F25	3	old	II.2	{31,35,45,50}
F25	4	very old	II.4	{48,55}

Figure 5 Metadata

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

Performance analysis and comparative study

In this section present some performance analysis results and then we compare the proposed storing solution to storing in a pure relational database model.

Performance analysis

In this section, we provide some performance analysis. First, we mention that the experimentations have been conducted on a Dell Vostro 1015 Laptop with an Intel Core 2 Duo processor (2.20 GHz) and 3 GB of memory. In addition, in these experimentations data have not been preloaded in memory and there is not any index on the attributes concerned by the insertion of new tuples.

We studied the CPU times for query processing and the membership degrees computing with different number of attributes. Figure 6.a represents graphically the evolution of the CPU time for query processing with crisp and fuzzy attributes. We may distinguish three zones in this figure: (i) for a low number of attributes (less than 10), the CPU time of query processing with fuzzy attributes is relatively higher than the CPU time of query processing with crisp attributes; (ii) for a moderate number of attributes (between 20 and 40), the CPU time of query processing with crisp and fuzzy attributes are almost the same; and (iii) for a high number of attributes (more than 40), the CPU time of query processing with crisp attributes varies nearly linearly while the CPU time of query processing with fuzzy attributes varies smoothly.

Figure 6.b shows graphically the variation of the CPU time of query processing and membership degrees computing with crisp attributes. As shown in this figure, the CPU time for computing the degree of membership is about 20ms for a low number of

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

attributes. Then, the CPU time of membership degrees computing varies almost linearly with the number of attributes.

Figure 6.c illustrates graphically the CPU time for query processing and membership degrees computing with crisp and fuzzy attributes. As we notice in this figure, the CPU time for query processing and membership degrees computing with crisp and fuzzy attributes remain nearly the same for small number of attributes (less than 10). However, the CPU time of membership degrees computing increases more quickly for a higher number of attributes (more than 10).

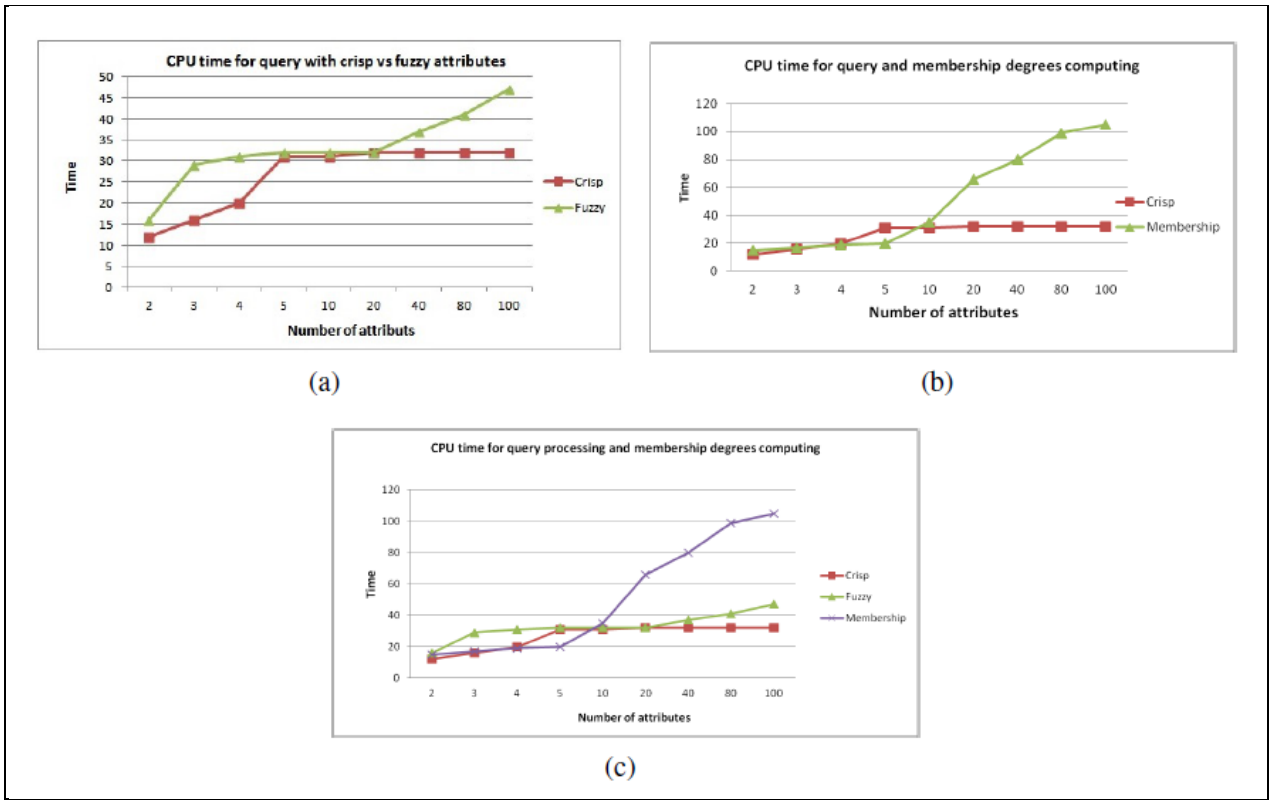


Figure 6 Performance analysis

We can conclude that querying do not suffer a big overhead when using fuzzy attributes and that the cost vary linearly with respect to the number of attributes.

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

However, the fact that in sometimes, it is faster to insert fuzzy attributes than crisp ones needs more investigations.

Comparative study

Figure 7 shows an extract from meta-relation FUZZ-ATTRIBUTES as defined in this paper and in a pure relational data model. It is easy to see that the use of multi-valued attributes, supported by object-relational database models, reduces the storage space and improves access time. The same remark holds for data storage as it is shown in Figure 8.

AttrID	AttrName	...	FuzzyDomain
1	Age	...	{0.65,1.7,2,2.5}
2	Location	...	{0.025,7.3}
3	Luminosity	...	{4.6}

AttrID	AttrName	...	Parameter1	Parameter2	Parameter3	Parameter4
1	Age	...	0.65	1.7	2	2.5
2	Location	...	0.025	7.3	nil	nil
3	Luminosity	...	4.6	nil	nil	nil

Figure 7 Extract from meta-relation FUZZ-ATTRIBUTES in our model (up) and in a pure relational database model (down)

TypeStar	...	Age Value	| DataType	|ParametersList	...	DOM
NOVA	...	Young	Linguistic Label	{0.7,1.6,2.1,3}	...	0.75
SUPERNOVA	...	10.2	Real	{}	...	1.0
SUPERNOVA	...	Very young	Interval	{0.01,0.85}	...	0.8

TypeStar	...	Age	DataType	Parameter1	Parameter2	Parameter3	Parameter4	...	DOM
NOVA	...	Young	Linguistic Label	0.7	1.6	2.1	3	...	0.5
SUPERNOVA	...	10.2	Real	nil	nil	nil	nil	...	1.0
SUPERNOVA	...	Very young	Interval	0.01	0.85	nil	nil	...	0.8

Figure 8 Extract from fuzzy relation STAR in our model (up) and in a pure relational database model (down)

[Post print version, please cite as] Sabrine Jandoubi, Afef Bahri, Nadia Yacoubi Ayadi, Salem Chakhar & Ashraf Labib (2016): Modelling, representation and implementation of imperfect information for an enhanced exploration of large databases, Journal of Decision Systems.
To link to this article: http://dx.doi.org/10.1080/12460125.2017.1252232

Discussion

In this section, we discuss some implementation issues. The first issue concerns the mapping and storing of fuzzy attributes' parameters. There are several solutions to map the characteristics of fuzzy attributes. We can, for example, use one common meta-relation with four attributes devoted to store the different parameters. In that time, we may have 'null' values any time the number of parameters is less than the maximum number of parameters. Another solution is to group data types along the number of required parameters. An ameliorated version of this solution is adopted in Medina et al (1995) where a common meta-relation is defined with a specific attribute serves as a pointer to two other meta-relations. One drawback of the solutions cited above is that anytime we need to add a new linguistic data type or to change the adopted linguistic data type, we may have to update the meta-relations structure.

The second issue concerns the use of the attributes' parameters both at the intensional and extensional levels. This allows users to insert values of different data types, which may have different number of parameters. For instance, the formal definition of the attribute may be a trapezoidal-based possibility distribution with four parameters but the user may introduce a crisp value (with no parameter), an interval (with two parameters) or an approximate value (with three parameters). In all cases, the different data types defined at the extensional level should be consistent with the formal definition of the attribute at the intensional level.

Conclusion

In this paper we proposed a set of conceptual and technical solutions to model, represent and implement imperfect information in the context of large fuzzy databases. The conceptual solution consists of a rich set of generic fuzzy data types permitting to model

almost all kinds of imperfect information. In addition, these fuzzy data types are uniformly represented using possibility distributions. Furthermore, the straightforward implementation solutions introduced in this paper exploit the basic constructs of object modelling such as multi-valued attributes, composed attributes, structured data types and array data types. The proposed solutions are particularly useful for exploring large databases since they minimize the space required to store imperfect information, and permit an efficient and effective access to these databases.

In the future, we first intend to use the advanced form of CREATE TYPE command to create Fuzzy ADT with all the required functionalities such as input and output functions. We also intend to apply the proposed solutions to a real-world decision problem requiring the use of large fuzzy databases and to conduct a series of intensive experimentations for evaluating the performance of the proposed solutions.

References

Bahri A., Chakhar S., Naijia Y., Bouaziz R., "Implementing imperfect information in fuzzy databases", 2nd International Symposium on Computational Intelligence and Intelligent Informatics, Hammamet, Tunisia, 290-294, 2005.

Berzal F., MarínN., Pons O., Vila M.A., "Managing fuzziness on conventional object-oriented platforms", International Journal of Intelligent Systems, 22(7):781-803, 2007.

Bosc W., Prade H., "An introduction to fuzzy set and possibility theory based approaches to the treatment of uncertainty and imprecision in database management systems", 2ndWorkshop on Uncertainty Management in Information Systems: From Needs to Solutions, 1990.

Bosc P., Kraft D., Petry F. , "Fuzzy sets in database and information systems: Status and opportunities", Fuzzy Sets and Systems, 156(3), 418-426, 2005.

Bouaziz R., Chakhar S., MousseauV., Ram S., Telmoudi A., "Database design and querying within the fuzzy semantic model", Information Sciences, 177(21):4598-4620, 2007.

Cadenas J.T., Marín N., Vila M.A., "Fuzzy domains with adaptable semantics in an object-relational DBMS", In Flexible Query Answering Systems, pp. 497-508. Springer Berlin Heidelberg, 2011.

Cadenas J.T., Marin N., Vila M.A., "Context-aware fuzzy databases", Applied Soft Computing, 25: 215-233, 2014.

Cuevas L., Marín N., Pons O., Vila M.A., "pg4DB: A fuzzy object-relational system", Fuzzy Sets and Systems, 159(12), 1500-1514, 2008.

Jandoubi S., Bahri A., Yacoubi-Ayadi N., Chakhar S., "Mapping the fuzzy semantic model into fuzzy object relational database model", The 7th International Conference on Information, Process, and Knowledge Management (eKnow 2015), Lisbon, Portugal, 138-143, 2015a.

Jandoubi S., Bahri A., Yacoubi-Ayadi N., Chakhar S., Labib A., "Enhanced fuzzy object-relational database model for efficient implementation of the fuzzy semantic model", The 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2015), Istanbul, Turkey, August 2-5, 2015b.

Klir G., Yuan B., "Fuzzy sets and fuzzy logic: Theory and applications", Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

Ma Z.M., "A conceptual design methodology for fuzzy relational databases", Journal of Database Management, 16(2):66-83, 2005.

Ma Z.M., Yan L., "A literature overview of fuzzy conceptual data modeling", Journal of Information Science and Engineering, 26(2)427-441, 2010.

Ma Z.M., Zhang F., Yan L., Cheng J., "Fuzzy data models and formal descriptions", Studies in Fuzziness and Soft Computing, 306: 33-60, 2014

Medina J.M., Vila M., Cubero J., Pons O., "Towards the implementation of a generalized fuzzy relational database model", Fuzzy Sets and Systems, 75(3):273-289, 1995.

Medina J.M., Pons J.E., Barranco C.D., Pons O., "A fuzzy temporal object-relational database: Model and implementation", International Journal of Intelligent Systems, 29(9:836-863, 2014.

Motro A., "Imprecision and incompleteness in relational databases: survey", Information and Software Technology, 32(9):579-588, 1990.

Rodrigues R.D., Cruz A., Cavalcanter R.T., "Alianca: A proposal for a fuzzy database architecture incorporating XML", Fuzzy Sets and Systems, 160(2):269-279, 2009.

Singh S., Agarwal K., Ahmad J., "Conceptual modeling in fuzzy object-oriented databases using unified modeling language", International Journal of Applied Engineering and Technology, 4(2):139-147, 2014.

Yan L., Ma Z.M., "information in fuzzy extended entity-relationship model and fuzzy relational databases", Journal of Intelligent and Fuzzy Systems, 27(4): 1881-1896, 2014a.

Yan L., Ma Z.M., "Modeling fuzzy information in fuzzy extended entity-relationship model and fuzzy relational databases", Journal of Intelligent & Fuzzy Systems: Applications in Engineering and Technology, 27(4):1881-1896, 2014b.

Yazici A., Buckles B., Petry F., "Handling complex and uncertain information in the ExIFO and NF^2 data models", IEEE Transactions on Fuzzy Systems, 7(6):659-676, 1999.