Control and Cybernetics

vol. **38** (2009) No. 2

Attribute-oriented defuzzification of fuzzy database tuples with categorical entries*

by

Rafal A. Angryk

Department of Computer Science
Montana State University
Bozeman, MT 59717-3880, USA
e-mail: angryk@cs.montana.edu

Abstract: We are investigating the ability to data mine fuzzy tuples, which are often utilized to represent uncertainty about the registered information. We discuss different aspects of fuzzy databases and comment on practical advantages of the model we utilized in our research. Motivated by a well known technique called Attribute-Oriented Induction, which has been developed for summarization of ordinary relational databases, we propose a new heuristic algorithm, allowing attribute-oriented defuzzification of fuzzy database tuples to the form acceptable for many regular (i.e. atomic values based) data mining algorithms. Significant advantages of our approach to defuzzification of fuzzy database tuples include: (1) its intuitive character of fuzzy tuples' interpretation, (2) a unique capability of incorporating background knowledge, implicitly stored in the fuzzy database models in the form of fuzzy similarity relation, directly into the imprecise data interpretation process, (3) transformation of fuzzy tuples to a format easy to process by regular data mining algorithms, and (4) a good scalability for time-efficient treatment of large datasets containing non-atomic, categorical data entries.

Keywords: defuzzification of fuzzy tuples, non-atomic categorical data entries, attribute-oriented induction, fuzzy databases, fuzzy similarity relation.

1. Introduction

There has been a very dynamic growth of data mining research during the last decade (Kantardzic and Zurada, 2005) leading to increased number of data mining algorithms and real-life applications. Many data analysis techniques

*Submitted: April 2008; Accepted: March 2009.

are being actively investigated with a focus on creating new and faster algorithms entirely devoted to time-efficient mining of massive data repositories. This change of research focus, from an accurate reality modeling and an intuitive querying to the fast large-scale data analysis, carries significant importance for all researchers working in areas related to fuzzy databases. Successful future of unconventional database models becomes more and more dependent on the development of consistent, easy to understand, and scalable mechanisms for mining the data such databases are capable of storing and processing. In future decades, providing effective mechanisms of imprecise data representation may be not enough to make the database models successful if no algorithms allowing for fast analysis of such data are available.

In this work we will address the challenge of transferring the fuzzy tuples (i.e. defuzzifying them) to forms acceptable for many regular (i.e. atomic values based) data mining algorithms. We decided to put our efforts into this task when we observed that non-atomic, categorical data entries, that can be so conveniently modeled in fuzzy databases, appear in large number of real-life database systems, ranging from multiple online surveys about customers' preferences (e.g. *mark types of food you like, in order of your preference*), to reports from medical examinations (e.g. *identify areas of pain, describe its severity*).

At the very beginning, we want to clarify a few important aspects of our work. First of all, we want to emphasize that our goal was not to develop a new technique for retrieving precise information about individual fuzzy tuples but instead—we focused on an algorithm allowing for a fast discovery of general trends in the entire data set. We also want to point out that our algorithm has been designed for defuzzification of fuzzy tuples containing categorical data, and extending it to the form allowing interpretation of continuous numeric values will require some modifications.

Secondly, it needs to be noted that the term "defuzzification", when used in this paper, should be interpreted in a much broader context than typically used in the fuzzy sets community (i.e. transformation of a single fuzzy set to a singleton). In this paper we will present a heuristics enabling us to transfer a set of fuzzy database tuples to the collection containing only atomic entries (i.e. singletons), that can be then easily interpreted by many regular data mining algorithms. These singletons, however, are intended for a discovery of general trends in the massive set of fuzzy data, not for a retrieval of precise information about an individual fuzzy tuple.

Finally, we want to accentuate that the point of this paper is not to argue about the statistical accuracy of the approach that has been used, but rather to show that a defuzzification of non-atomic data stored in fuzzy databases is practically achievable, and to raise awareness in the fuzzy database community about the lack of algorithms that can quickly transfer data stored in large fuzzy databases to a form that can be easily analyzed with popular data mining software (e.g. Weka, 2009).

In the next section, we present an overview of the extended fuzzy relational database model and briefly comment on these aspects of the model, which we consider important from data mining practitioners' perspective. We also give a brief review of research conducted on attribute-oriented induction—a technique which has motivated our approach to fuzzy databases defuzzification. In Section 3, we propose a new heuristic algorithm to interpret fuzzy tuples and then present some experimental results we obtained, when examining behavior of our algorithm. Finally, in Section 4, we summarize the things we learned from our investigation.

2. Related works and motivation

2.1. Extended fuzzy relational database model

In this work we are focusing on the extended fuzzy relational database model, with the shape of database tables matching an example presented in Table 1. Following early fuzzy database models (Buckles and Petry, 1982, 1983; Petry, 1996), we will start with allowing each of the descriptive attributes to store more than just an atomic descriptor. In other words, for each actual attribute entry, denoted as a_{ij} (where j is an index of the attribute, and i is the number of the tuple in the fuzzy database table), we let $a_{ij} = \{d_j : d_j \in D_j,\ 0 \leq |a_{ij}| \leq |D_j|\}$ (i.e. we extend an atomic entry requirement: $a_{ij} = \{d_j : d_j \in D_j,\ |a_{ij}| \leq 1\}$ coming from Codd's, 1970, First Normal Form), where D_j denotes the domain of the j^{th} attribute (denoted as A_j), and d_j is its singleton element. Now, following Prade and Testemale (1984), we further extend the attribute characterization to the format allowing to store information about the possibility of each entered value occurrence (i.e. we extend each acceptable attribute entry to fuzzy subset of this attribute domain): $a_{ij} = \{(d_j|\mu_{ij}(d_j)) : d_j \in D_j,\ \mu_{ij}(d_j) \mapsto (0, 1],\ 0 \leq |a_{ij}| \leq |D_j|\}$. In the example presented in Table 1, $Nickname_{123} = \{Ivan|1.0, Big|0.6\}$ means that there is a possibility equal to 1.0 that the person identified using number 123 is called *Ivan*, and a possibility equal 0.6 that he is referred to as *Big*. This notation has been used in the past (Buckles and Petry, 1982, 1983; Petry, 1996; Prade and Testemale, 1984; Baldwin and Zhou, 1984; Raju and Majumdar, 1988; Shenoi and Melton, 1989; Zemankova-Leech and Kandel, 1984; Rundensteiner, Hawkes and Bandler, 1989; Zadeh, 1970; Medina, Pons and Vila, 1994) to reflect an *imprecision* of the registered attribute value(s). Non-atomic entries to the fuzzy attribute (i.e. the $d_j|\mu_{ij}(d_j)$ pairs) can be interpreted (Urrutia et al., 2006) either as *inconsistent*—i.e. logical *XOR* (only one value is correct, but we were not sure which one, so both values were entered), or in a truly multi-valued manner, that is, as a *conjunction* of terms (where only the merger of entered values is relevant, e.g. *bird* that is covered in *white* and *black* feathers), or as a *disjunction* (where all possible combinations are acceptable, e.g. *bird* that may be all *black*, or all *brown*, or even carry both colors). In our algorithm we assume *disjunctive* in-

terpretation of non-atomic data entries. For example, the suspect with *Id*=123 (Table 1) is a citizen of three different countries: *Ukraine, Belarus* and *USA* (and may be also considered as a member of a small group of suspects who take advantage of multiple citizenships).

Table 1. Fuzzy database table containing human trafficking suspects

Id	***Nickname***	***Country of citizenship***	***Country of operation***	***Suspicion of involvment***
123	*{Ivan|1.0, Big|0.6}*	*{Ukraine|0.8, Belarus|0.9, USA|0.6}*	*{USA|1.0}*	*0.6*
245	*{Rookie|1.0}*	*{USA|1.0}*	*{USA|0.8, Colombia|0.3}*	*0.8*
987	*{Boss|1.0}*	*{Colombia|0.9, Venezuela|0.8, Spain|0.4}*	*{Colombia|1.0*	*0.1*

To make sure our fuzzy database model reflects extended fuzzy database capabilities, we included two additional properties often discussed in the literature on fuzzy databases (Buckles and Petry, 1982, 1983; Petry, 1996; Prade and Testemale, 1984; Baldwin and Zhou, 1984; Raju and Majumdar, 1988; Shenoi and Melton, 1989; Zemankova-Leech and Kandel, 1984; Rundensteiner, Hawkes and Bandler, 1989). First, to make sure that our model is compatible with works presented in Prade and Testemale (1984), Baldwin and Zhou (1984), Raju and Majumdar (1988), we added an attribute with membership function $\chi_R(t_i) \mapsto (0, 1]$ to each fuzzy database table. Function $\chi_R(t_i)$ reflects the degree to which a given fuzzy tuple t_i belongs to a particular fuzzy database relation (the relation is denoted by index R). It is used to model *uncertainty* (Prade and Testemale, 1984; Medina, Pons and Vila, 1994) about the belonging of an individual tuple to the particular database table. For example, the last column in Table 1 shows that $\chi_{HumanTraffickingSuspect}(t_{123}) = 0.6$, which can be interpreted as 60% degree of suspicion/possibility that someone using nicknames *Ivan* and *Big* is involved in the business of trading human beings. It needs to be mentioned that this attribute has a slightly different role than the other attributes. Whereas the focus of the remaining attributes (we will call them *descriptive*) is on characterization of the registered entity *per se* (and fuzzy sets are used to reflect *imprecision* of this description), the last attribute characterizes strength of (or *uncertainty* about, Prade and Testemale, 1984; Medina, Pons and Vila, 1994) the entity membership (i.e. relevance) to the particular database table.

Moreover, as in Buckles and Petry (1982, 1983), Petry (1996), Shenoi and Melton (1989), Zemankova-Leech and Kandel (1984), Rundensteiner, Hawkes and Bandler (1989), we let each of the *descriptive* domains have a pre-specified

binary fuzzy relation, which extends a crisp equivalence relation regularly used in ordinary databases. Some of the early works on fuzzy databases (Buckles and Petry, 1982, 1983; Petry, 1996) used Zadeh's fuzzy similarity relation (Zadeh, 1970), whereas more recent models (Shenoi and Melton, 1989; Rundensteiner, Hawkes and Bandler, 1989) seemed to prefer using fuzzy proximity (Shenoi and Melton, 1989) or resemblance (Rundensteiner, Hawkes and Bandler, 1989) relations (which both behave in a similar manner). Being mostly interested in practicality and scalability of our model, when used for analysis of large repositories containing categorical (i.e. non-continuous) data entries, we decided to make a compromise and employ a technique which quickly transforms proximity and resemblance relations to the format identical with the Zadeh's similarity relation. However, before explaining our transformation formally, we would like to comment briefly on crucial differences between the above-mentioned fuzzy relations. All these fuzzy relations are reflexive and symmetric (Angryk, 2006), but what distinguishes Zadeh's similarity relation from the other two is its max-min transitivity. This is a very important property from the perspective of our algorithm, as it leads to nested character of fuzzy similarity classes (where the classes from the higher level of abstraction contain one or more classes from the lower level), which speeds up significantly real-life data comparisons. Moreover, there are means of transferring a fuzzy proximity relation to the fuzzy similarity relation, which have been shown in Shenoi and Melton (1989).

For instance, let us consider Table 2, which represents a fuzzy proximity relation specified for the domain *Country*. The table can be transformed to the max-min transitive fuzzy similarity relation (the results of the transformation are presented in Table 3) using Tamura's proximity chains (Tamura, Higuchi and Tanaka, 1971). For instance, despite the fact that the proximity degree between the concepts *Lithuania* and *Spain* is 0.2 (presented in Table 2), our similarity degree, derived from this proximity table using Tamura's sequences, is 0.4 (Table 2). The result was obtained using the sequence (chain) of the original proximity degrees (Table 2): $Prox_{Country}(Lithuania,\ Belarus) = 0.7 \wedge Prox_{Country}(Belarus,\ Ukraine) = 0.7 \wedge Prox_{Country}(Ukraine,\ Spain) = 0.4$, we generated Tamura's chain of proximities (i.e. fuzzy conjunction based on operator MIN) with the weakest link equal to 0.4, obtaining the result of $Sim_{Country}(Lithuania,\ Spain) = 0.4$, as presented in Table 3.

More formally (Shenoi and Melton, 1989), if $Prox_{D_j}$ is a proximity relation, specified by experts for domain D_j (e.g. Table 2), then given an $\alpha \in [0, 1]$, two values from this domain $d_k, d_l \in D_j$ are said to be *α-proximate* if, and only if, $Prox_{D_j}(d_k, d_l) \geq \alpha$. We will say that these two attribute values are α-similar if, and only if, we can find a set of domain elements $d_1, d_2, \ldots, d_m \in D_j$, such that a sequence of $Prox_{D_j}(d_k, d_1) \wedge \bigcap_{i=1}^{m-1} Prox_{D_j}(d_i, d_{i+1}) \wedge Prox_{D_j}(d_m, d_l) \geq \alpha$ (i.e. fuzzy conjunction of α-proximities) can be derived. We derived such sequence for the pair *Lithuania* and *Spain* above, and the entire similarity relation for our example is presented in Table 3.

Table 2. Proximity relation for domain *Country*, with non-convex 0.4-proximity classes marked in grey

	Lithuania	*Belarus*	*Ukraine*	*Spain*	*Portugal*	*Canada*	*USA*	*Colombia*	*Venezuela*
Lithuania	1.0	0.7	0.6	0.2	0.1	0.0	0.0	0.0	0.0
Belarus	0.7	1.0	0.7	0.3	0.2	0.0	0.0	0.0	0.0
Ukraine	0.6	0.7	1.0	0.4	0.4	0.0	0.0	0.0	0.0
Spain	0.2	0.3	0.4	1.0	0.7	0.0	0.0	0.0	0.0
Portugal	0.1	0.2	0.4	0.7	1.0	0.0	0.0	0.0	0.0
Canada	0.0	0.0	0.0	0.0	0.0	1.0	0.7	0.3	0.2
USA	0.0	0.0	0.0	0.0	0.0	0.7	1.0	0.4	0.2
Colombia	0.0	0.0	0.0	0.0	0.0	0.3	0.4	1.0	0.7
Venezuela	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.7	1.0

Table 3. Similarity relation for domain *Country*, derived from Table 2

	Lithuania	*Belarus*	*Ukraine*	*Spain*	*Portugal*	*Canada*	*USA*	*Colombia*	*Venezuela*
Lithuania	1.0	0.7	0.7	0.4	0.4	0.0	0.0	0.0	0.0
Belarus	0.7	1.0	0.7	0.4	0.4	0.0	0.0	0.0	0.0
Ukraine	0.7	0.7	1.0	0.4	0.4	0.0	0.0	0.0	0.0
Spain	0.4	0.4	0.4	1.0	0.7	0.0	0.0	0.0	0.0
Portugal	0.4	0.4	0.4	0.7	1.0	0.0	0.0	0.0	0.0
Canada	0.0	0.0	0.0	0.0	0.0	1.0	0.7	0.4	0.4
USA	0.0	0.0	0.0	0.0	0.0	0.7	1.0	0.4	0.4
Colombia	0.0	0.0	0.0	0.0	0.0	0.4	0.4	1.0	0.7
Venezuela	0.0	0.0	0.0	0.0	0.0	0.4	0.4	0.7	1.0

Now, the disjoint classes of attribute values, which are considered to be equivalent at a specific α-level, can be extracted from each proximity relation. The α-similarity classes (also referred to as *equivalence classes*, or *α-cuts on similarity relation*, Zadeh, 1970) are marked in Table 3 with different shadings for different values of α. The transformation, which converts the original proximity table to a fuzzy similarity relation, has tremendous practical use. The practical advantage of the proximity-based approach comes from the lack of necessity to worry about the max-min transitivity when defining the proximity

degrees. This makes proximity tables much easier for human experts to define. In practice however, non-convex α-proximity classes (e.g. see 0.4-cuts on proximity relation in Table 2) are hard to interpret and sometimes can generate significant computational costs during data processing. Using sequences of overlapping α-proximate classes, we can now dynamically generate fuzzy similarity relations that exclusively characterize similarities between only these attribute values, which were actually entered into the particular fuzzy database table. They will have a nested character (as marked by shadings in Table 3, and also shown as groupings in Fig. 1), which significantly speeds up processing of large data sets. Obviously, the transitive property (Zadeh, 1970; Tamura, Higuchi and Tanaka, 1971) of Zadeh's similarity relation carries some limitations, especially for data containing continuous values without clearly separated clusters, and therefore the transformation of proximity relation to similarity relation should be executed with caution (we recommend paying special attention to distribution of data values, especially in cases where the actual data does not show well-separated clusters). It seems useful to also mention here that although the proximity tables are usually defined by human experts, we can also automatically extract similarity relations using data-driven techniques. For instance, a Hierarchical Agglomerative Clustering (HAC) algorithm (Han and Kamber, 2006), commonly used to generate hierarchies of non-overlapping (at individual levels) clusters, provides a structure which maintains all properties of Zadeh's partition tree. Speaking about dendrograms, we recommend review of recent literature on some interesting attempts to flatten these structures (Chuang and Chien, 2002, 2004; Wall, Richter and Angryk, 2005), to make them more human-friendly and also more practical to use when processing large data sets. It is a well known fact that usually expert-specified partition trees have bushier (and therefore more flat) character, in contrast to HAC-derived hierarchies of clusters, which generate very tall, binary trees.

The existence of a fuzzy similarity relation for each *descriptive* attribute A_j of a fuzzy database table allows us to extract a partition tree (identical to that presented by Zadeh, 1970, representing disjoint equivalence classes on individual α levels (see Fig. 1). From the propagation of shadings in Table 3, we can observe that the equivalence classes are separated (at individual level of partition tree) and have a nested character (Zadeh, 1970)—i.e. values joined into a single similarity class at one level of partition tree can never be separated at the higher levels of abstraction. This is very useful in practice—clearly separated equivalence classes are easier to interpret, and the trees are far more time-efficient to traverse (Angryk and Petry, 2005, 2007).

Utilization of Zadeh's partition trees (where the classes from the higher level of abstraction nest one or more classes from the lower level), also called concept hierarchies, has proven to be very useful in many data cube-related operations implemented in data warehouses (e.g. roll-up, drill-down commands, Han and Kamber, 2006). We will talk a little bit more about use of concept hierarchies in data mining applications in the next section (2.2).

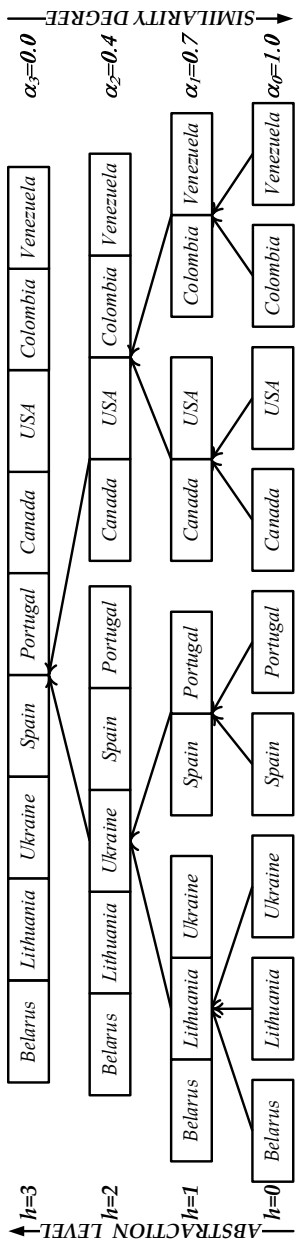


Figure 1. Disjoint α-equivalence classes on multiple levels of Partition Tree for Similarity Relation from Table 3

Before going further with our presentation, we would like to introduce a formal notation that we are going to use when referring to partition trees. A partition tree (example presented in Fig. 1), defined for a single domain in fuzzy database model, can be interpreted as a set of interconnected ordered pairs (i.e. domain's abstraction levels) denoted as G_j, such that $G_j = \{(D_j^h, \alpha_h) : 0 \leq h < H_j,\ \alpha_h \in [0,1]\}$, where D_j^h denotes domain of categorical attribute A_j on h^{th} abstraction level, with $\left|D_j^h\right| > \left|D_j^{h+1}\right|$ (the granularity increase property is caused by the self-nested character of equivalence classes on different levels of the partition tree), and $\alpha_h > \alpha_{h+1}$ (the similarity decrease property is caused by the abstraction increase within the tree). In our definition, $D_j^h = \left\{d_{j1}^h, d_{j2}^h, \ldots, d_{j|D_j^h|}^h\right\}$ is the set of equivalence classes defined for the h^{th} abstraction level, where d_{jm}^h denotes the m^{th} node (i.e. equivalence class) at the h^{th} level of the partition tree G_j. For instance, a set of equivalence classes at α_1=0.7 level in Fig. 1 (denoted as set of $D^1_{Country}$) consists of four nodes: $\{d_1^1 = \{$*Belarus, Lithuania, Ukraine*$\}$, $d_2^1 = \{$*Spain, Portugal*$\}$, $d_3^1 = \{$*Canada, USA*$\}$, $d_4^1 = \{$*Colombia, Venezuela*$\}\}$. In this context, we can narrow our definition of fuzzy *descriptive* attribute to the form: $a_{ij} = \{(d_{jm}^0 | \mu_{ij}(d_{jm}^0)) : d_{jm}^0 \in D_j^0,\ \mu_{ij}(d_{jm}^0) \mapsto (0,1]\}$.

To summarize this section, we want to point out three important features our fuzzy database can register: (1) *imprecision* of the entity description (reflected by the fuzziness of information entered into its *descriptive* attributes), (2) *similarity* of the observed characteristics (preserved in domain-specific similarity relations specified for the *descriptive* attributes), and (3) relevance of the registered entity to the particular topic/relation (reflected by the degree of *uncertainty*, denoted by us with $\chi_R(t_i)$ symbol and stored in the last attribute of fuzzy database table).

2.2. Attribute-oriented induction

Data generalization is a process of grouping of data, enabling transformation of similar entries, stored originally in a database at the low (primitive) level, into more abstract conceptual representations. This process is a fundamental element of Attribute-Oriented Induction (AOI) (Han and Kamber, 2006; Han, Cai and Cercone, 1992 and 1993) – a descriptive database mining technique, allowing compression of the original data set into a *generalized relation*, which provides a concise summary of the massive set of original data. Process of gradual data summarization is also used widely in data warehouses, where data cubes are often generalized (using roll-up operations, Han and Kamber, 2006) to abstraction levels preferred by the data analysts.

AOI progressively transforms the original, massive set of data (i.e. *initial relation*) into a concise form at the higher abstraction level, called a *generalized relation*. Generalization of database tuples is performed on an attribute-by-

attribute basis, applying a separate concept hierarchy for each of the generalized attributes included in the relation of task-relevant data. Each concept hierarchy represents background knowledge about the attribute domain, allowing gradual aggregation of attribute values stored in the original database tuples.

The concept hierarchies, which are necessary to permit gradual, similarity-based, aggregation of attribute values, are usually built bottom-up, progressively increasing the abstraction of the generalization concepts at each new level. Creation of new concept levels in generalization hierarchies is accompanied by an increase of the concept abstraction (i.e. our h in Fig. 1) and the decrease of number of concepts, which mirrors the $\alpha_h > \alpha_{h+1}$ and $\left|D_j^h\right| > \left|D_j^{h+1}\right|$ properties of Zadeh's (1970) partition trees.

A brief example of AOI (Han and Kamber, 2006) is presented in Tables 4 and 5. Assume we have retrieved a data table from the university database (depicted in Table 4). This table is our *initial relation* and can be extracted from the original database using a regular relational database query.

Table 4. Example of initial relation (Han and Kambler, 2006)

Name	***Gender***	***Major***	***Birth Place***	***Birth Date***	***Gpa***
J.Smith	*M*	*CS*	*Vancouver, BC*	*08/12/'88*	*3.67*
S.McSon	*M*	*EE*	*Montreal, QE*	*07/28/'85*	*3.71*
L.Node	*F*	*Physics*	*Seattle, WA*	*03/25/'98*	*3.93*
...	...	...	...	...	...

Table 5. Example of generalized relation (Han and Kambler, 2006)

Gender	***Major***	***Birth Region***	***Age Range***	***Gpa***	***Count***
M	*Engineering*	*Canada*	*20-25*	*V. good*	*2*
F	*Science*	*USA*	*25-30*	*Excellent*	*1*
...	...	...	...	...	...

By utilizing AOI we are able to compress the retrieved data into a more general form, as illustrated in Table 5. This table is the output of an AOI algorithm (i.e. a *generalized relation*), where some or all attributes are generalized and counts (or other representations of tuples aggregation) are accumulated.

Table 5 represents general characterization of the students' population. At the achieved level of abstraction we did not preserve names of individual students and many other details were abandoned. Nevertheless, the performed AOI generated a concise summary, which has useful and informative character

and is also much faster to process by many data mining algorithms (due to its reduced size). The table might be used to quickly learn which majors are the most popular among students, how many students are females, where they come from, etc. A new attribute *Count* was added to the generalized relation. It allows preservation of information about proportions among the tuples originally stored in the university database.

Before going further with our investigation, we want to point out an obvious connection between attributes *Count* in the generalized relation (Table 5) and *Suspicion of involvement* in our fuzzy database example (Table 1). Whereas all tuples in the regular databases completely belong to the data table, the $\chi_R(t_i)$ function represents a degree of this belonging in fuzzy databases. In such a case, it seems to be natural that adding a fuzzy tuple t_i to the output table should cause increase of *Count* attribute not necessarily by 1, but rather by an appropriate value of $\chi_R(t_i)$.

Depending on the approach and the intention of data analysts, generalization of collected data can be treated either as a final step of data mining (e.g. summary-tables are presented to decision makers, allowing them to interpret overall information, Angryk and Petry, 2007; Han, Cai and Cercone, 1992) or as an introduction to further knowledge extraction (e.g. extraction of abstract association rules directly from the generalized data, Angryk and Petry, 2007; Han, Cai and Cercone, 1993). For readers interested in alternative approaches to summarization of databases, we would also like to recommend the works of Kacprzyk and Zadrozny (2005), and Yager and Petry (2006).

3. Attribute-oriented defuzzification of fuzzy tuples

In this work, our goal is to precisely interpret fuzzy tuples (i.e. to defuzzify them). The AOI concept of using *<generalized tuple in 1NF, count of original data tuples>* pair provided us with motivation for our work on attribute-oriented defuzzification. Our goal, however, is not necessarily to generalize a fuzzy tuple, but rather to transfer it into a collection of tuple fractions of the following format: *<list of descriptive attributes in 1NF, its vote>*. In other words, we want to specialize (i.e. defuzzify) the fuzzy tuple into a collection of tuples containing only atomic values (i.e. equivalence classes at the 0^{th} abstraction level), followed by a number (i.e. *vote*) reflecting its importance. We used the term *vote* here, just to distinguish it from the *count*, mentioned above, although essentially they play almost the same role (i.e. preservation of original proportions among the data entries). Being the result of aggregation of multiple tuples, the term *count* always represents a natural number ($count \in \mathfrak{I}^+$), whereas in our defuzzification approach we will allow the *vote* to reflect a fraction of one, imprecise tuple ($vote \in \mathfrak{R}^+$).

Although we are not interested in generalization of fuzzy data, we still want to transfer our fuzzy data tables (e.g. Table 1) into the forms compatible with Han's *generalized data tables* (e.g. Table 5). The AOI output tables are in the

1NF (Codd, 1970) and therefore can be easily analyzed by many popular data mining algorithms (Kantardzic and Zurada, 2005; Weka 3, no date; Angryk, 2006; Han and Kamber, 2006).

Having background knowledge available in the form of fuzzy similarity relations, we were also naturally interested in making sure that our defuzzification mechanism takes advantage of this information. The fuzzy similarity relations provide us with useful and important knowledge of experts that needs to be utilized in data mining applications, as it may allow for more accurate interpretation of imprecise data.

Before going further with our investigation, we would like to emphasize that our approach is intended for mining general patterns in large repositories of imprecise data (e.g. discovery of countries that may be involved in human trafficking), and should not be used as a "precise" querying technique (e.g. trying to decide if the suspect named *Ivan* comes from *Ukraine* or *Belarus*). We are not proposing here a solution for precise interpretation of individual fuzzy tuples, but rather focusing on discovery of general knowledge, despite the fact that the gathered data set contains imprecision.

3.1. On fuzzy tuple defuzzification

Before presenting implementation details of our algorithm (Table 8), we would like to discuss it using an example. Let us look at the last tuple of the fuzzy database table presented in Table 1. What *is* (or *are*, if the person has multiple citizenships) the citizenship of the suspect identified via *Id*=987 (i.e. the *Boss*), who (as not-confirmed reports say) had been using passports from {*Colombia*|0.9, *Venezuela*|0.8, *Spain*|0.4}? And, even more importantly, how crucial is he for our investigation purposes in context of all the data we gathered?

The latter question seems to be easier to answer than the former one. Clearly, the degree of belongingness, denoted in section 2.1 via $\chi_R(t_i) \mapsto (0, 1]$ and placed in the last column of Table 1, should have direct influence on the importance of the fuzzy tuple in the generated 1NF output table. In ordinary databases, each tuple fully belongs to the data table in which it is stored (we can say: $\forall\, t_i \in R$ $\chi_R(t_i) = 1$), so the *Count* of the generalized tuple is incremented by 1, each time a new tuple is added to the generalized tuple. In the case of the fuzzy databases, however, the importance (belongingness) of each tuple is explicitly denoted using the membership function $\chi_R(t_i)$. Therefore, all fractions of a fuzzy tuple t_i, which are generated by our defuzzification algorithm, need to be weighted using its $\chi_R(t_i)$ value at the end of the defuzzification process. This can be easily achieved by multiplying a vote of each fraction of a defuzzified tuple by its $\chi_R(t_i)$ value before adding it to the output relation. Moreover, it seems reasonable to expect that imprecision of the entries stored in individual descriptive attributes would also have some influence on the importance (i.e. *vote*) of the particular fuzzy tuple, when compared with other (maybe less imprecise and more relevant) tuples, stored in our data table.

We will discuss these topics in detail shortly, but now let us go back to the earlier question. How can we map (i.e. defuzzify) a descriptive attribute entry, represented by the fuzzy set $\{Colombia|0.9, Venezuela|0.8, Spain|0.4\}$, into an atomic form? This question opens a broad range of interesting possibilities.

We could simply use MAX operator to defuzzify the $\{Colombia|0.9, Venezuela|0.8, Spain|0.4\}$ entry, with rationale that we want to follow the most possible option (i.e. we transform original imprecise entry into atomic one, choosing the one with maximal possibility of occurrence, in our case: $\{Colombia\}$, and disregarding information about the possibility of its occurrence). Usage of MAX operator is very effective computationally (i.e. fast), however, it causes a complete removal of the whole imprecision, which we wanted to be able to register in the first place.

Another approach is to try to split the vote equally among all descriptors inserted by the user. We could say that $\frac{1}{3}$ of the fuzzy tuple vote (assume for now that we talk about a tuple with only one fuzzy attribute) should be assigned to reflect atomic value *Colombia*, $\frac{1}{3}$ to *Venezuela*, and $\frac{1}{3}$ to *Spain*. We split the original tuple evenly and transfer it to three 1NF tuples, each carrying one-third of the vote. This technique ensures that the information about all entered descriptors is reflected in the output table. The approach seems to be more reasonable than the earlier one; however its disadvantages are still significant. It does not take into consideration important real-life information, stored in our imprecise attribute and reflected not only by the number of inserted descriptors (d^0_{jm}'s), but also by (1) the information about possibility of their occurrence in the given a_{ij} (reflected via $\mu_{ij}(d^0_{jm})$'s, which have been entered by the user), and (2) the similarity of the entered d^0_{jm}'s (reflected by the similarity relation G_j defined by domain experts). The expert knowledge, reflected in similarity relation contains important information that may help us with interpretation of imprecise information (e.g. the knowledge that countries *Belarus* and *Ukraine* are more similar to each other than to the *USA*, changes the way we are looking at the citizenship of *Ivan* in Table 1, making us think that the suspect may be of Eastern European origin).

It is important to emphasize that we want to reflect in the output table the original proportions between all pieces of information, which we managed to save in our fuzzy data table.

Han and Kamber (2006), Han, Cai and Cercone (1992, 1993) used *Count* to make sure all original relations between the database tuples are preserved at the AOI output. Our task is more complex, since we need to assure that our output table maintains information about: (1) relevance of a fuzzy tuple (i.e. its *uncertainty*), reflected explicitly by $\chi_R(t_i)$, and (2) its *imprecision*, reflected by the number of entered categorical values (d^0_{jm}'s) and their degrees of possibility ($\mu_{ij}(d^0_{jm})$'s), stored in its *descriptive* attributes. We plan to achieve this by making sure each descriptive attribute of a fuzzy tuple maintains equal weight of unity during all internal (i.e. *attribute-specific*) steps of the tuple defuzzification.

This even representation of tuple descriptive attributes is maintained until the final step of defuzzification, when the importance (i.e. relevance) of all the tuple fractions can be weighted using its (i.e. *tuple-specific*) $\chi_R(t_i)$value, right before adding it to the other tuples in the output (i.e. defuzzified) data table.

This rationale suggests a natural order of the defuzzification process. First, we want to defuzzify all (one after another) descriptive attributes of the fuzzy tuple, then weight its all fractions using the original $\chi_R(t_i)$ value. We will refer to the first phase as an *attribute-specific* part of defuzzification, since it needs to be performed on all descriptive attributes, which are fuzzy. The second phase applies to all fractions of the tuple and uses its original $\chi_R(t_i)$ value; therefore we named it a *tuple-specific* step of defuzzification.

In attribute-specific phase, we transfer our original fuzzy set

$$a_{ij} = \{Colombia|0.9, Venezuela|0.8, Spain|0.4\}$$

to the format where individual possibility values (denoted as $\mu_{ij}(d^0_{jm})$) are normalized based on the total of their original values (i.e. $\sum_{d^0_{jm} \in a_{ij}} \mu_{ij}(d^0_{jm})$). Our normalized entry could have the following form:

$$an_{ij} = \{(d^0_{jm}|\mu^n_{ij}(d^0_{jm})) : d^0_{jm} \in a_{ij}, \mu^n_{ij}(d^0_{jm}) = \frac{\mu_{ij}(d^0_{jm})}{\sum_{d^0_{jm} \in a_{ij}} \mu_{ij}(d^0_{jm})}\},$$

and we could treat our new μ^n_{ij}'s as fractions of the attribute-specific vote: $\sum_{d^0_{jm} \in a_{ij}} \mu^n_{ij}(d^0_{jm}) = 1$ (this is the reason why we normalized our $\mu^n_{ij}(d^0_{jm})$ by the sum of related $\mu_{ij}(d^0_{jm})$'s).

Thus, in our example, first we need to add all original $\mu_i j$'s (i.e. $\sum_{d^0_{jm} \in a_{ij}} \mu_{ij}(d^0_{jm})$ $= 0.9 + 0.8 + 0.4 = 2.1$) to obtain a normalizing divisor, then we can normalize individual μ_{ij}'s so they would reflect attribute-specific distribution of the tuple vote. We can now transform our fuzzy attribute *aij* value into three fractions: *Colombia* – with $\frac{0.9}{2.1} = 0.43$ of the attribute-specific vote (i.e. $\mu^n_{ij}(Colombia)$), *Venezuela* with $\frac{0.8}{2.1} = 0.38$ of the vote, and the last descriptor *Spain* with $\frac{0.4}{2.1} = 0.19$ of the vote. This distribution reflects better the possibility values entered by the user than the even (into one-thirds) split of the attribute-specific vote.

This solution, however, still does not reflect all information we have about the attribute domain, which may be especially useful when interpreting highly imprecise attribute entries (i.e. when more than two descriptors were entered into a fuzzy set a_{ij}). The current distribution of possibilities (μ^n_{ij}'s) reflects only information about the imprecision the user wanted to register when inserting multiple descriptors, but does not carry any information from the expert

knowledge we have about these values (i.e. that the countries *Colombia* and *Venezuela* are much more similar to each other than to *Spain*, and therefore we should probably pay more attention to the first two descriptors).

To incorporate background knowledge, we propose further modifications to the distribution of our attribute-specific vote fractions. Our goal is to make sure that these fractions also reflect the similarity of the inserted descriptors (d^0_{jm}'s). Using the partition tree (e.g. Fig. 1), we can distinguish from the set of the originally inserted values these concepts which are more similar to each other than to the remaining attribute values and adjust distribution of imprecise tuple fractions in such a way that they match more accurately the background knowledge provided by experts. We call the descriptors, which are more similar to each other than to other inserted values, the *subsets of resemblance* (e.g. *Colombia, Venezuela*} from the above example). We can use them as a basis for calculating a more intuitive distribution of the vote fractions in the case of highly imprecise entries (i.e. when we have the number of descriptors bigger than two). An important aspect of our approach is the extraction of the *subsets of resemblance* at the lowest possible abstraction level of their common occurrence, since the nested character of equivalence classes in partition trees guarantees that above this level they are going to co-occur regularly.

Our heuristic algorithm for incorporation of fuzzy similarity relation into defuzzification of imprecise attributes is quite straightforward and we present it formally later (Table 8). For now, assume that you are given (1) a set of domain values inserted as a description of a particular entity attribute (e.g. in our case: {*Colombia, Venezuela, Spain*}), and (2) a tree-like structure reflecting a partition tree for the particular attribute (e.g. Fig. 1). Using these inputs, we want to extract a table, which includes (1) the list of all *subsets of resemblance* occurring in the given fuzzy tuple, and (2) the highest degree of similarity (i.e. α-level in the partition tree) of their common occurrence (see Table 6). The reason that we decided to use only the highest value of α is caused by the repetitive occurrences of the same subsets at the multiple levels of the partition tree. This could result in unbalancing the original dependencies among inserted values, skewing significantly the results of our similarity-based defuzzification toward *subsets of resemblance* occurring at the very low levels of partition trees. Now, we can use the highest degrees of similarity (shown in Table 6) to intuitively distribute fractions of the original tuple.

Our algorithm uses depth-first recursive traversal of a partition tree when searching for *subsets of resemblance*. The partition tree is searched starting from its root and, if any subset of descriptors characterizing the given fuzzy tuple attribute occurs at the particular node of the partition tree, we store the values that were recognized as α–similar and the adequate degree of similarity (α). This degree may get its value updated as the attribute values continue to co-occur in the common equivalence classes, when we continue to traverse appropriate subtrees of the partition tree. We present an example of such a search for subsets of resemblance in a descriptive attribute containing the earlier mentioned values

{*Colombia, Venezuela, Spain*} in Fig. 2. Numbers on the edges in the tree represent the order in which the partition tree was traversed and individual *subsets of resemblance* were discovered (or updated). *Subsets of resemblance* generated for this example are presented in Table 6.

Table 6. Subsets of resemblance for the analyzed example

Subsets of resemblance at their highest similarity levels	***COMMENTS***
{Colombia, Venezuela, Spain} | 0.0	*STORED*
{Spain} | 0.4	*STORED*
{Spain } | ~~0.4~~ 0.7	*UPDATED*
{Spain } | ~~0.7~~ 1.0	*UPDATED*
{Colombia, Venezuela} | 0.4	*STORED*
{Colombia, Venezuela} | ~~0.4~~ 0.7	*UPDATED*
{Colombia} | 1.0	*STORED*
{Venezuela} | 1.0	*STORED*

After extracting the *subsets of resemblance*, we apply a simple summarization of their α values as a measure reflecting both the frequency of occurrence of the individual attribute values in the α-similarity classes, as well as the abstraction level of these occurrences. Since the country of *Spain* was found only twice in the *subsets of resemblance*, we assigned it a grade 1.0 (i.e. 1.0+0.0). The remaining descriptors were graded as follows: $Colombia|(1.0 + 0.7 + 0.0) = Colombia|1.7$, $Venezuela|(1.0 + 0.7 + 0.0) = Venezuela|1.7$.

In the next step, we add all generated grades ($1.0 + 1.7 + 1.7 = 4.4$) to normalize the output grades assigned to each of the participating attribute values in exactly the same way we did earlier: $Spain|\frac{1.0}{4.4} = 0.227$, $Colombia|\frac{1.7}{4.4} = 0.386$, $Venezuela|\frac{1.7}{4.4} = 0.386$. Since this fuzzy set is derived from a similarity relation, we denote it as as_{ij} and describe it as a collection of pairs: $(d^0_{jm}|\mu^s_{ij}(d^0_{jm}))$ such that $d^0_{jm} \in a_{ij}, \mu^s_{ij}(d^0_{jm}) \mapsto (0, 1]$, where $\sum_{d^0_{jm} \in a_{ij}} \mu^s_{ij}(d^0_{jm}) = 1$ (again!) and the individual μ^s_{ij}'s reflect distribution of relevant equivalence classes in the pre-specified fuzzy similarity relation.

It is important to realize that this similarity-based defuzzification mechanism shows its advantage only when more than two descriptors have been entered into a fuzzy attribute. Since the fuzzy similarity relation is symmetric, there is no point in trying to use this mechanism to defuzzify fuzzy set a_{ij} that consists of only two elements – it will always generate $\mu^s_{ij}(d^0_{jm}) = 0.5$ for both of the descriptors. Nevertheless, we strongly believe that the similarity-based defuzzification shows its advantages when we need to deal with highly imprecise data entries, as it allows us to incorporate knowledge of experts into interpretation

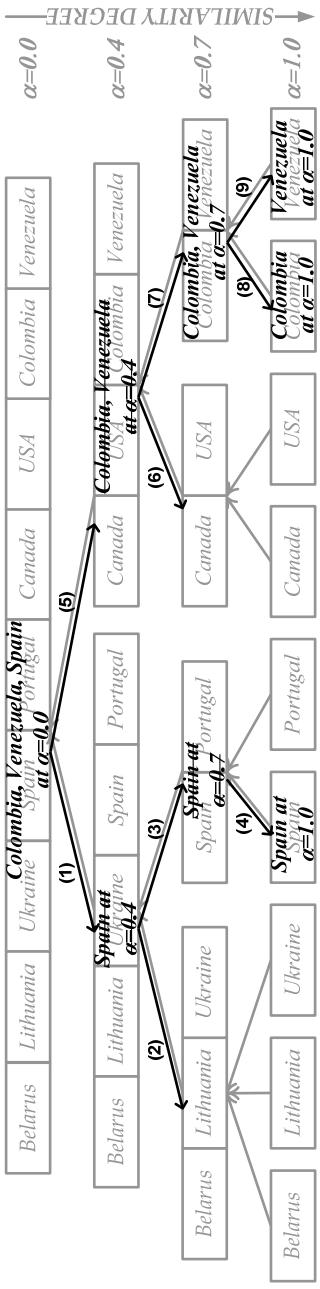


Figure 2. Subsets of resemblance extracted from the partition tree of Fig. 1

of such data. Instead of treating all categorical values identically, we can take advantage of existing background knowledge about the similarity of the entered values.

Now, we are left with the task of linking the knowledge from the user (who entered the descriptors and possibilities of their occurrences), and from the expert (who developed similarity table characterizing relations between these descriptors). Distribution of memberships, normalized using the user's observation, suggested: *Colombia*|0.43 of the attribute-specific vote, *Venezuela*|0.38, and *Spain*|0.19, while our similarity-based algorithm generated: *Colombia*|0.386, *Venezuela*|0.386, and *Spain*|0.227.

Since both sets of results sum their membership values to unity (each), we can quickly merge them using the following formula: $\forall\, d^0_{jm} \in a_{ij}$ update the pair $(d^0_{jm}|\mu_{ij}(d^0_{jm}))$ with $\mu_{ij}(d^0_{jm}) = \frac{\mu^n_{ij}(d^0_{jm})+\mu^s_{ij}(d^0_{jm})}{2}$. Now, the distribution of μ_{ij}'s ads up to unity ($\sum_{d^0_{jm}\in a_{ij}} \mu_{ij}(d^0_{jm}) = 1$) and it incorporates both sources of knowledge (i.e. the user and the experts): *Colombia*$|\frac{0.43+0.386}{2} = 0.408$, *Venezuela*$|\frac{0.38+0.386}{2} = 0.383$, and *Spain*$|\frac{0.19+0.227}{2} = 0.209$. Obviously, this average-based method of merging our partial results is quite simplified. For purposes of preliminary investigation, we decided to treat both sides (i.e. the user, who entered the data, and the expert, who developed the similarity table) equally. A more sophisticated approach (e.g. weighted average) may be a better fit in some real-life applications.

In this way we can maintain the desired property of all descriptive attributes maintaining even weight (i.e. 1) during attribute-specific steps of our defuzzification process. Moreover, the current distribution of membership values reflects all information stored in our descriptive attribute. That is: (1) the number of entered descriptors (reflecting user's doubt about the observed characteristics), (2) possibility of their occurrence (reflecting user's conviction about some of the descriptors being more accurate than others), and finally (3) the background knowledge about the investigated domain (reflected in the form of fuzzy similarity relation). This leads to the new distribution of the fuzzy attribute fractions, which more accurately reflects real-life dependencies than a linear weighting approach.

This attribute-specific normalization of membership values (μ_{ij}'s) can be repeated for all descriptive attributes which contain fuzzy entries. If we assume that a fuzzy tuple t_i consists of N descriptive (and fuzzy) attributes a_{i1}, a_{i2}, ..., a_{iN}, followed by an additional attribute, denoted as $a_{i(N+1)}$, which contains its $\chi_R(t_i)$ value, we can defuzzify t_i by generating the cross product of all descriptors stored in t_i's descriptive attributes (i.e. $\{d^0_{1m} : d^0_{1m} \in a_{i1}\} \times \{d^0_{2m} : d^0_{2m} \in a_{i2}\} \times \ldots \times \{d^0_{Nm} : d^0_{Nm} \in a_{iN}\}$), and then weighting each of the 1NF tuples (i.e. fractions of the fuzzy tuple t_i), generated by our cross product operation, by the result of multiplication of the related $\mu_{ij}(d^0_{jm})$'s and the original $\chi_R(t_i)$ value. This weight represents the *vote* of the given *ti*'s fraction.

In other words, by defuzzification of a fuzzy tuple t_i, we mean decomposition of t_i's descriptive attributes (represented using fuzzy sets) into a collection of tuples (i.e. fractions of the fuzzy tuple) with their descriptive attributes containing only atomic entries (i.e. d^0_{jm}'s that were originally entered into the fuzzy tuple), extended with an additional attribute *vote*, which is generated by multiplying all normalized possibility values (i.e. relevant $\mu_{ij}(d^0_{jm})$'s), and the $\chi_R(t_i)$ value.

Formally, if we denote a fuzzy database tuple as

$$t_i =< a_{i1}, a_{i2}, \ldots, a_{iN}, a_{i(N+1)} >$$

such that for $j = 1...N$ is a descriptive attribute containing a fuzzy set

$$\{(d^0_{jm}|\mu_{ij}(d^0_{jm})) : d^0_{jm} \in D^0_j, \mu_{ij}(d^0_{jm}) \mapsto (0, 1]\},$$

where $\mu_{ij}(d^0_{jm})$ is normalized using techniques presented above, and $a_{i(N+1)} = \chi_R(t_i)$, then we can defuzzify t_i by transferring it to a set of 1NF tuples (i.e. fractions of t_i), denoted as F_i, where each of F_i's elements has the following format: $f_{ik} =< f_{k1}, f_{k2}, ..., f_{kN}, f_{k(N+1)} >$ such that for $j = 1...N$ $f_{kj} = d^0_{jm}$ where $d^0_{jm} \in a_{ij}$, $f_{k(N+1)} = a_{i(N+1)} \cdot \prod_{j=1}^{N} \mu_{ij}(f_{kj})$, and $k = 1... \prod_{j=1}^{N} |a_{ij}|$. $|a_{ij}|$ denotes the cardinality of the fuzzy set.

Example of a defuzzified table (using Table 1 as input and assuming that the attribute *Nickname* has been removed before defuzzification) is presented in Table 7. To further reduce the size of the output data table, we can remove attribute *Id* and merge the tuples which have the same values in all descriptive attributes (e.g. rows 3 and 4 in Table 7). To maintain original proportions among the data entries when merging these tuples, we need to add their votes. The resulting relation has a format very similar to the generalized relation presented in Table 5 and can be easily analyzed by popular data mining algorithms.

Implementation details of our attribute-oriented defuzzification algorithm are presented in Table 8. As the input, we assume: (1) a fuzzy tuple, denoted as t_i, with the total number of descriptive attributes equal N, and $\chi_R(t_i)$ specified as the $(N+1)^{th}$ attribute, and (2) a set of N *Partition Trees*, reflecting fuzzy similarity relations specified for descriptive attributes and denoted as $G_{1...N}$. On the output, we get a collection of database tuples in the 1NF format (each denoted as f_{ik} reflecting the fuzzy tuple t_i. Each of the output tuples has the following format: N atomic attribute values $d^0_{k1}, d^0_{k2}, ..., d^0_{kN}$ (such that $d^0_{kj} \in a_{ij}$), followed by a *vote* (i.e. $(N+1)^{st}$ attribute) representing how strongly the original fuzzy tuple t_i's is reflected in the particular f_{ik} tuple. In Table 8 we denoted the whole set of output tuples (i.e. t_i's fractions) as F_i. The set reflects the complete defuzzified (i.e. atomic) representation of an individual fuzzy tuple t_i.

When describing a fuzzy tuple processing, we used a few temporary data structures to make our pseudo-code easier to read. The most important ones are specified in the Internal Data Structures section of Table 8. To save space in

Table 7. Defuzzified database table, derived from Table 1

Id*	***Country of citizenship***	***attribute-specific vote****	***Country of operation***	***attribute-spec. vote****	***Vote***
123	*Ukraine*	$(\frac{0.8}{0.8+0.9+0.6}+\frac{1.0+0.7+0.0}{1.7+1.7+1.0}):2=0.367$	*USA*	1.0	$0.367\cdot 1.0\cdot 0.6=0.22$
123	*Belarus*	$(\frac{0.9}{0.8+0.9+0.6}+\frac{1.0+0.7+0.0}{1.7+1.7+1.0}):2=0.389$	*USA*	1.0	$0.389\cdot 1.0\cdot 0.6=0.233$
123	*USA*	$(\frac{0.6}{0.8+0.9+0.6}+\frac{1.0+0.0}{1.7+1.7+1.0}):2=0.244$	*USA*	1.0	$0.244\cdot 1.0\cdot 0.6=0.146$
245	*USA*	1.0	*USA*	$\frac{0.8}{0.8+0.3}=0.727$	$1.0\cdot 0.727\cdot 0.8=0.582$
245	*USA*	1.0	*Colombia*	$\frac{0.3}{0.8+0.3}=0.273$	$1.0\cdot 0.273\cdot 0.8=0.218$
987	*Colombia*	$(\frac{0.9}{0.9+0.8+0.4}+\frac{1.0+0.7+0.0}{1.7+1.7+1.0}):2=0.408$	*Colombia*	1.0	$0.408\cdot 1.0\cdot 0.1=0.041$
987	*Venezuela*	$(\frac{0.8}{0.9+0.8+0.4}+\frac{1.0+0.7+0.0}{4.4}):2=0.383$	*Colombia*	1.0	$0.383\cdot 1.0\cdot 0.1=0.038$
987	*Spain*	$(\frac{0.4}{0.9+0.8+0.4}+\frac{1.0+0.0}{1.7+1.7+1.0}):2=0.209$	*Colombia*	1.0	$0.209\cdot 1.0\cdot 0.1=0.021$

* *The attribute ID and all calculations have been shown only to make our presentation more understandable. These results are not displayed in the output table generated by our algorithm.*

Table 8. Algorithm for attribute-oriented defuzzification of fuzzy database tuple

INPUT:

Fuzzy Tuple, denoted as t_i, containing $a_{i1}, a_{i2}, ..., a_{iN}$ descriptive attributes, and one additional attribute representing t_i's belonging to the fuzzy database table R : $a_{i(N+1)} = \chi_R(t_i)$.

Partition Trees for all descriptive attributes. Each of the trees is denoted as G_j, where $j = 1 \ldots N$, and $G_j = \left\{ (D_j^h, \alpha_h) : 0 \leq h < H_j, \alpha_h \in [0,\ 1] \right\}$, where our a_{ij} entry is a fuzzy set on appropriate D_j^0 and d_{jm}^h denotes the m^{th} node (i.e. equivalence class) in the h^{th} level of the tree G_j.

Internal Data Structures:

Results of a_{ij}'s normalization, based on the values of possibilities entered by the user, are stored in: $an_{ij} = \{(d_{jm}^0 | \mu_{ij}^n(d_{jm}^0)) : d_{jm}^0 \in D_j^0, \mu_{ij}^n(d_{jm}^0) \mapsto (0, 1]\}$ such that $\sum_{d_{jm}^0 \in a_{ij}} \mu_{ij}^n(d_{jm}^0) = 1$.

Results of a_{ij}'s normalization using fuzzy similarity relation are stored in: $as_{ij} = \{(d_{jm}^0 | \mu_{ij}^s(d_{jm}^0)) : d_{jm}^0 \in D_j^0, \mu_{ij}^s(d_{jm}^0) \mapsto (0, 1]\}$ such that $\sum_{d_{jm}^0 \in a_{ij}} \mu_{ij}^s(d_{jm}^0) = 1$.

Subset of Resemblance (see Table VI for an example) is a set of ordered pairs denoted as: $R_j = \{(r_p, \alpha_p) : r_p \subseteq d_{jm}^h, (d_{jm}^h, \alpha_p) \in G_j\}$, where $\alpha_p = MAX\{\alpha_h : (r_p, \alpha_h) \in G_j\}$.

OUTPUT:

Defuzzified Tuple, a set of t_i's fractions, denoted as F_i, such that each of F_i's elements is in 1NF and has the following format $< f_{k1}, f_{k2}, ..., f_{kN}, f_{k(N+1)} >$, where f_{kj} denotes a single equivalence class of the j^{th} attribute at the 0^{th} abstraction level (i.e. $f_{kj} \in D_j^0$), such that $f_{kj} \in a_{ij}$ for all descriptive attributes (i.e. $j = 1 \ldots N$), and $f_{k(N+1)} = \chi_R(t_i) \cdot \prod_{j=1}^{N} \mu_{ij}(f_{kj})$.

METHOD:

(1) $F_i = \{\emptyset\}$;
(2) **for** (**each** descriptive attribute a_{ij}, where$j = 1...N$) {
(3) **if** (a_{ij} contains non-atomic value) { $R_j = \{\emptyset\}$; $an_{ij} = \{\emptyset\}$; $as_{ij} = \{\emptyset\}$;
(4) **for** (**each** $(d_{jm}^0 | \mu_{ij}(d_{jm}^0)) \in a_{ij}$) { $\mu_{ij}^n(d_{jm}^0) = \frac{\mu_{ij}(d_{jm}^0)}{\sum_{d_{jm}^0 \in a_{ij}} \mu_{ij}(d_{jm}^0)}$;
$an_{ij} = an_{ij} \bigcup (d_{jm}^0 | \mu_{ij}^n(d_{jm}^0))$; }
(5) $a_{ij} = an_{ij}$;
(6) } // end of "if (a_{ij}contains non-atomic value)"
(7) **if** (a_{ij}contains more than 2 values) {
(8) Using all $d_{jm}^0 \in a_{ij}$, generate a list of all non-empty subsets of a_{ij}, which are ordered from the largest subset to the subsets containing only 1 element. The list is named **Candidates for Subsets of Resemblance**, and denoted as $C_j = \{c_q : c_q \subseteq a_{ij}, |c_q| \geq |c_{q+1}| \geq 1\}$;
(9) **for** (**each** equivalence class $d_{jm}^{H_j - 1}$ from the top of Gj) **DFSearchForResemblances**(d_{jm}^{H-1}, C_j);
(10) **for** (**each** $d_{jm}^0 \in a_{ij}$) { **for** (**each** pair (r_p, α_p)in Rj) **if** $(d_{jm}^0 \in r_p)$ $\mu_{ij}^s(d_{jm}^0) = \mu_{ij}^s(d_{jm}^0) + \alpha_p$;
(11) $as_{ij} = as_{ij} \bigcup (d_{jm}^0 | \mu_{ij}^s(d_{jm}^0))$; }
(12) **for** (**each** $(d_{jm}^0 | \mu_{ij}^s(d_{jm}^0)) \in as_{ij}$) $\mu_{ij}^s(d_{jm}^0) = \frac{\mu_{ij}^s(d_{jm}^0)}{\sum_{d_{jm}^0 \in as_{ij}} \mu_{ij}^s(d_{jm}^0)}$;
(13) **for** (**each** $d_{jm}^0 \in a_{ij}$) $\mu_{ij}(d_{jm}^0) = \frac{\mu_{ij}^n(d_{jm}^0) + \mu_{ij}^s(d_{jm}^0)}{2}$;
(14) } //end of "if (a_{ij}contains more than 2 values)"
(15) } //end of "for each descriptive attribute a_{ij}"
(16) **CreateDefuzzifiedTuples** (a_{i1}, 1);
(17) **return**;

```
procedure DFSearchForResemblances (Node of a Partition Tree d^h_jm, Candidates for Resemblances C_j)
(18)  if (h < 0 ∨ C_j == {∅}) return;
(19)  if (c_1 ⊆ d^h_jm) UpdatePairOfResemblances(c_1, α_h);
(20)  else { C_j = C_j\{c_1};
(21)      DFSearchForResemblances(d^h_jm, C_j); }
(22)  for (each d^{h-1}_jm) if (d^{h-1}_jm ∈ DirectDescendentsOf(d^h_jm)) DFSearchForResemblances(d^{h-1}_jn, C_j);
(23)  return;
procedure UpdatePairOfResemblances (Candidate for Resemblances c, Similarity Level α)
(24)  for (each pair (r_p, α_p) ∈ R_j) if (r_p == c) { α_p = α; return; }
(25)  R_j = R_j ⋃ (c, α);
(26)  return;
procedure CreateDefuzzifiedTuples (Attribute a_ij, Index of Fraction k)
(27)  if (j > N) {k = k + 1; f_{k(N+1)} = a_ij; return; }
(28)  for (each d^0_jm ∈ a_ij) { c = k + 1;
(29)      CreateDefuzzifiedTuples(a_{i(j+1)}, k);
(30)      for (ℓ = c...k) { f_ℓj = d^0_jm; f_{ℓ(N+1)} = f_{ℓ(N+1)} · μ_ij(f_ℓj); }
(31)  return;
```

memory, the final results of attribute-specific defuzzification (i.e. normalization using μ^n_{ij} and μ^s_{ij} values) are saved back to the original attribute (a_{ij}), before the tuple decomposition (using a cross product) and weighting (using the original $\chi_R(t_i)$ value) is performed in the recursive procedure named *CreateDefuzzifiedTuples*.

The algorithm takes a single fuzzy tuple as an input, so it needs to be run on all imprecise tuples in the fuzzy database table.

3.2. Experimental results

To carefully evaluate a large-scale performance of the defuzzification algorithm, we conducted our experiments using artificially generated datasets and different types of partition trees. To make sure our results can be clearly interpreted and are not influenced by the cost of merging a large number of fuzzy attributes, which we implemented in the recursive *CreateDefuzzifiedTuples* procedure (lines 27-31 in Table 8), our fuzzy data table contains only two attributes: a descriptive one (denoted as a_{i1}), and another attribute containing t_i relevance value ($a_{i2} = \chi_R(t_i)$).

The test data for the descriptive attribute $a_{i1}$was generated by randomly picking values from the domain containing 32 symbols, which were interpreted as distinct equivalence classes at the lowest (i.e. α=1.0) abstraction level of all partition trees we used (see Fig. 3). The a_{i1}entry was considered to carry 75% of imprecision (see Table 9 for examples), when the number of elements in this attribute was at 75% of the domain size at the lowest abstraction level (i.e. we had $|a_{i1}| = 24$ pairs of the format $(d^0_{1m}|\mu_{i1}(d^0_{1m}))$ in our descriptive attribute entry, with the domain containing $\left|D^0_1\right| = 32$ equivalence classes at the least abstract level). To make sure our data entries are random, but reflect the same imprecision, we have chosen 24 descriptors (i.e. our d^0_{1m}'s) in random order for

each tuple of the 75%-imprecise data set and merged them with randomly generated membership degrees (i.e. our $\mu_{i1}(d^0_{1m})$'s). The degrees of belongingness ($\chi_R(t_i)$), which we stored in the second attribute (a_{i2}), were also created using a random number generator. We made sure that all$\mu_{i1}(d^0_{1m})$ and $\chi_R(t_i)$ values fit into $(0, 1]$ range. It is important to note that the distribution of all values in our fuzzy tuples is completely random and does not follow distributions of equivalence classes in any of the partition trees, which we have used during our experiments.

To make sure we conduct comparable experiments, we decided to preserve the original randomness in our imprecise data sets when running all of our experiments. We initially generated one data file with 30,000 fuzzy tuples, where each of the tuples contained the descriptive attribute a_{i1} with a randomly generated fuzzy set representing 75% of data imprecision, followed by a random number from the (0,1] range, which simulated *ai2* entry. To generate the 71.875%-imprecise data file we simply eliminated one pair $(d^0_{1m}|\mu_{i1}(d^0_{1m}))$ from each fuzzy set a_{i1} stored in our originally 75%-imprecise data file. We continued this elimination process until three pairs remained in each a_{i1} entry, which is reflected as the $\frac{|a_{i1}|}{|D^0_1|} = \frac{3}{32} = 9.375\%$ of imprecision in our plots. For scalability evaluation, we have chosen two earlier generated data files (one with 12.5% and another one with 75% of data imprecision), and removed respective tuples from both data files, gradually generating derivative files with 28, 26, ..., 4, and finally 2 thousands of fuzzy tuples.

All of our data files were tested against different types of partition trees, reflecting different relations of fuzzy similarity among the categorical values occurring in our artificial domain of the attribute A_1. Fig. 3 presents six different fuzzy similarity relations $(G1, G2, \ldots, G6)$. We ordered them based on their numbers of abstraction levels. The same attribute domain (D^0_1) was used in different types of partition trees, as presented on the top of Fig. 3 ($G1$), although the symbols are not explicitly presented in the remaining hierarchies. To make things easier to compare - each of our fuzzy similarity relations presented in Fig. 3 has been derived from 32 distinct equivalence classes at the lowest abstraction level, and all trees have $\alpha = 1.0$ at the bottom, and $\alpha = 0.0$ at the top.

Please note that although partition trees $G4$ and $G5$ represent identical shapes, they have different α-values. $G5$ depicts a similarity relation where the highly similar concepts are quickly unified at the higher values of α, and $G4$ represents a case where symbolic values are not that similar and are unified at more abstract levels (i.e. with lower values of α). Although we had expected that running time of our algorithm was dependent on the shape of our partition trees, rather than on a distribution of α-values within an identically shaped tree, we generated these two similarity relations to experimentally confirm our expectations (see Figs. 4-7 for the results). $G4$ and $G5$ have a total of five abstraction levels in their hierarchy ($H = 5$). They have also the same average branching factor ($b = 2.148$).

Table 9. Examples of fuzzy records used during our tests

Fuzzy record $t_i = < a_{i1}, a_{i2} >$		**Data**
a_{i1}	a_{i2}	**Imprecision**
a|0.59, d|0.24, g|0.61, j|0.94, m|0.56, p|0.73, s|0.04, w|0.19, z|0.45, b|0.69, e|0.75, h|0.3, k|0.01, n|0.92, q|0.21, t|0.53, x|0.79, c|0.62, f|0.24, i|0.89, l|0.52, o|0.07, r|0.98, u|0.12	*0.63*	$\frac{\lvert a_{i1}\rvert}{\lvert D_1^0\rvert} = \frac{24}{32} = 75\%$
a|0.59, d|0.24, g|0.61, j|0.94, m|0.56, p|0.73, s|0.04, w|0.19, z|0.45, b|0.69, e|0.75, h|0.3, k|0.01, n|0.92, q|0.21, t|0.53, x|0.79, c|0.62, f|0.24, i|0.89, l|0.52, o|0.07, r|0.98	*0.12*	$\frac{\lvert a_{i1}\rvert}{\lvert D_1^0\rvert} = \frac{23}{32} = 71.875\%$
q|0.07, d|0.24, g|0.61, j|0.94, m|0.56, p|0.73, s|0.04, w|0.19, z|0.45, b|0.69, e|0.75, h|0.3, k|0.01, n|0.92, c|0.21, t|0.53, x|0.79, a|0.62, f|0.24, i|0.89, l|0.52, o|0.07	*0.47*	$\frac{\lvert a_{i1}\rvert}{\lvert D_1^0\rvert} = \frac{22}{32} = 68.75\%$

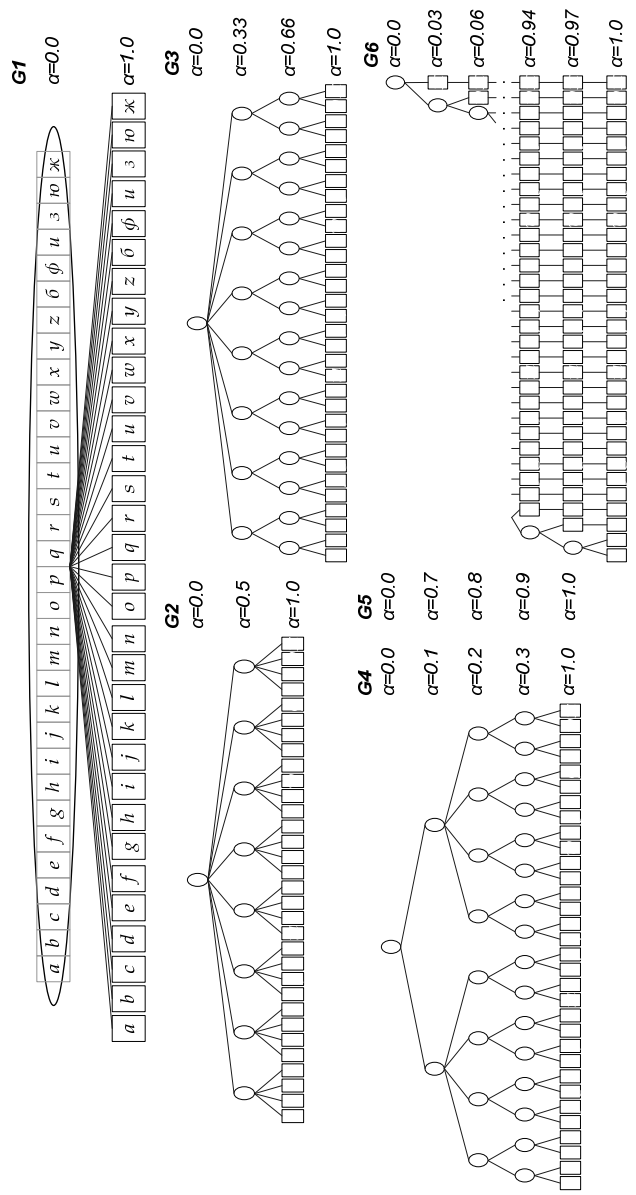


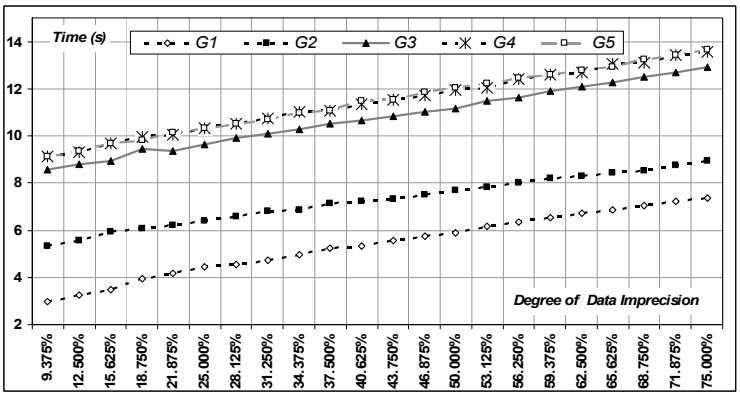
Figure 3. Different types of partition trees for descriptive attribute a_{i1} used during our tests ($|D_1^0| = 32$).

$G1$ and $G6$ represent the extreme cases. The hierarchy $G1$ has only the root and the leaves—its average branching factor is $b = 32$ and the value of H is equal 2 ($b >> H$), while the hierarchy $G6$ is a classical dendrogram with the average branching factor b being equal to 1.0625, and the height of $H = 32$ ($b << H$). Some levels of $G6$ have been omitted in Fig. 3 due to the limited space of this presentation. Obviously, $G6$ represents the tallest tree we could consider for the domain of 32 values.

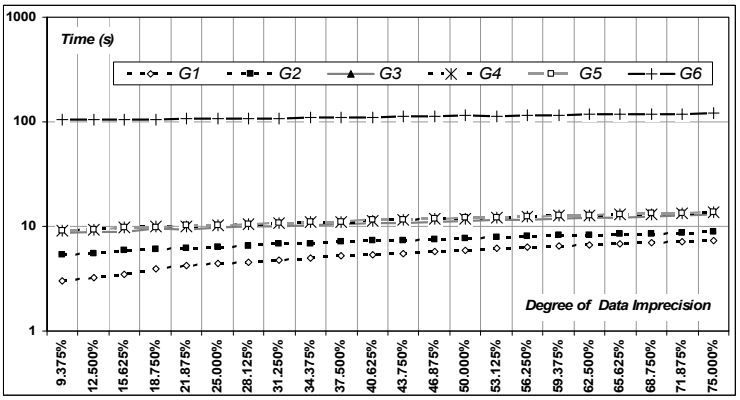
Our algorithm has been tested using different types of hierarchies from Fig. 3 and data sets with different percentages of data imprecision (see Fig. 4 for results), and with different numbers of equally imprecise tuples (scalability tests are presented in Figs. 5-7). Results presented in Fig. 4, show that time required to run our data defuzzification algorithm is linearly proportional to the degree of imprecision occurring in the fuzzy attribute for the fuzzy data tables with imprecision = 75%. The running time for $G1$ is the best because it has the least number of levels, which gives it an advantage since our data defuzzification algorithm uses a depth-first search (see *DFSearchForResemblances* procedure in Table 8). Fig. 4 also contains timings for two look-alike hierarchies, $G4$ and $G5$, which have identical shapes but different distributions of α-values. It is evident from Fig. 4 (a) (see two overlapping plots on the top), that the running times for these two hierarchies are the same. This is because our data defuzzification algorithm traverses the same nodes, while storing and updating matches found in the hierarchies with different values of α. It is important to note that the vote fractions, generated when traversing these two trees, may be distributed differently due to differences in propagation of α's in each of the trees. The overlapping plots in Fig. 4 confirm, however, that the distribution of α-values among the levels does not have a direct impact on running time of our data defuzzification algorithm. Moreover – the running time of the algorithm seems to be linearly correlated with the degree of data imprecision, and should be expected to grow with the number of levels in the partition tree (i.e. our H).

Figs. 4(b) and 5 show that the tallest hierarchy (G_6) takes the maximum time compared to all other hierarchies. From our experiment, it becomes apparent that the procedure *DFSearchForResemblances* (Table 8) conducts a number of level-dependent searches, where the performance decreases with increases in the number of levels in the partition tree, which suggests that flattening of the partition trees (i.e. making them more bushy, similarly to the way B+trees are arranged) in large-scale applications may be a recommended approach (Chuang and Chien, 2002, 2004; Wall, Richter and Angryk, 2005). It is important to remember, however, that our test results were generated with randomly distributed data points. In real life we would expect some kind of positive correlation between distribution of imprecise attribute entries in a fuzzy data table and a fuzzy similarity relation defined for the descriptive attribute.

A higher degree of data imprecision, in the case of partition trees with a very low branching factor b (see Figs. 5, and 4(b) for the results generated by $G6$, which has $b = 1.0625$), causes our *DFSearchForResemblances* procedure



(a)

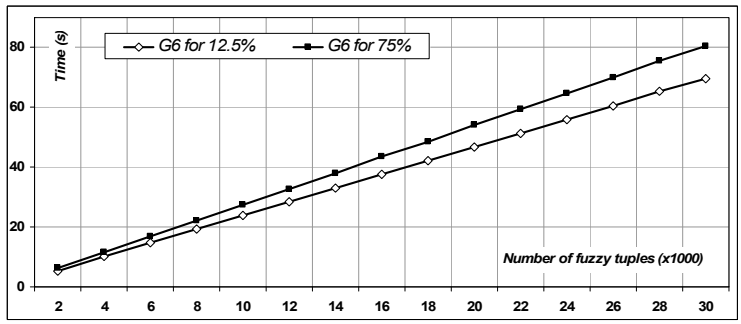


(b)

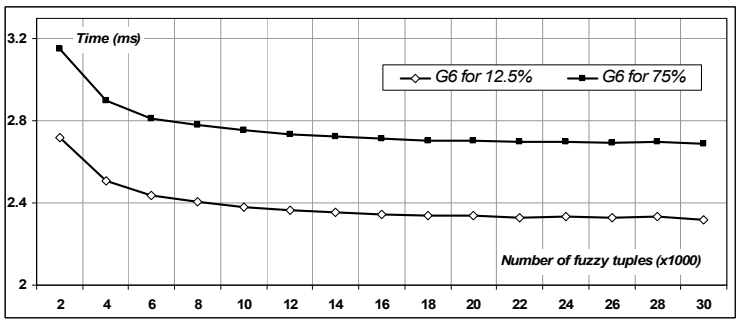
Figure 4. Total running time as a function of data imprecision for different shapes of partition trees (number of data records for all plots = 30,000

to traverse further before the benefits of the branch-pruning strategy (see lines 18-23 in Table 8), can be noticed. This is why tall trees with a low branching factor b generate higher cost when used against the data sets with very high imprecision (i.e. 75% in our examples). Our results show well the costs that may be necessary to interpret large sets of highly imprecise data. They also confirm our expectation that flattening (i.e. increasing of an average branching factor) of dendrogram-like hierarchies, before using them for data generalization or specialization, can be very useful in practice.

Figs. 5-7 depict the running time as a function dependent on the number of tuples in the dataset (with the degree of data imprecision unchanged). We carried out our scalability tests using two different degrees of data imprecision:



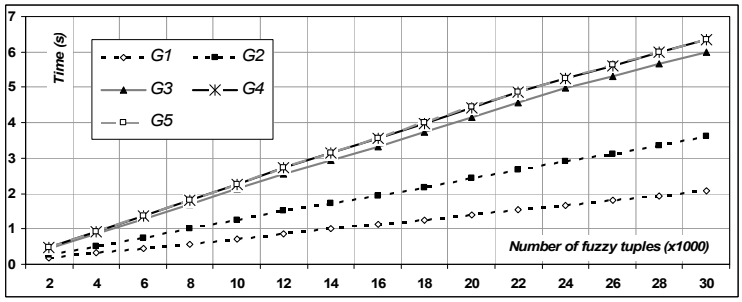
(a)



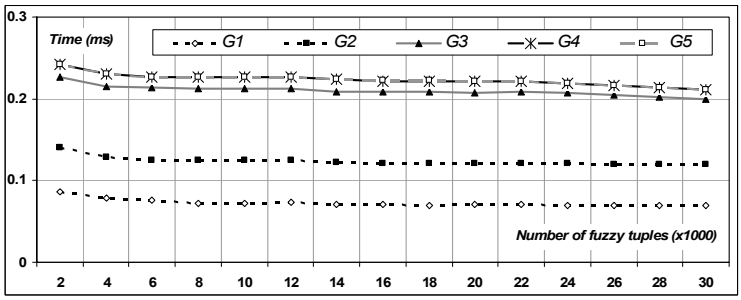
(b)

Figure 5. The worst case scenario — algorithm scalability for dendrogram-like partition trees (partition tree $G6$ in Fig. 3). (a) Actual time taken by our algorithm. (b) Average time to traverse a single record

12.5% and 75%. Figs. 6 and 7 show that our defuzzification algorithm has a linear performance with respect to the number of fuzzy tuples, when run using the five examples of partition trees we introduced earlier (i.e. $G1 - G5$ in Fig. 3). Since $G6$ has the highest number of levels, it takes the longest time to traverse when compared with flatter trees (the scalability tests for dendrogram are presented in Fig. 5). On the other hand, $G1$ has only two levels and therefore it took the shortest time to check using our recursive *DFSearchForResemblances* procedure at every corresponding number of fuzzy tuples. Fig. 6(b) shows the average time to traverse a partition tree for a single tuple with the same data as used to generate results presented in Fig. 6(a). The same relation occurs between parts (a) and (b) in Figs. 5 and 7. In Fig. 6(b) and 7(b), average timing lines are almost straight and become parallel to the horizontal axis as the number of tuples increases. As expected, it can be noticed that our algorithm constant initiation time resulted in the average time of defuzzification per tuple being slightly higher for low numbers of tuples in our fuzzy data table (as shown



(a)

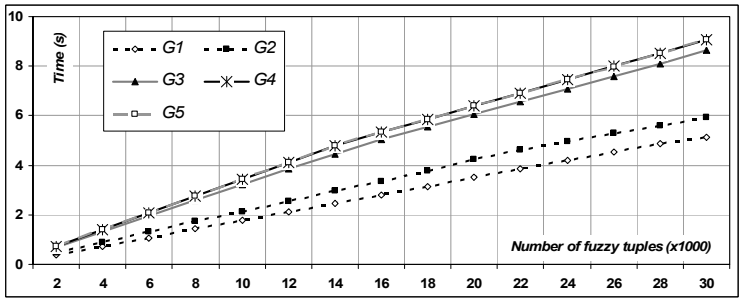


(b)

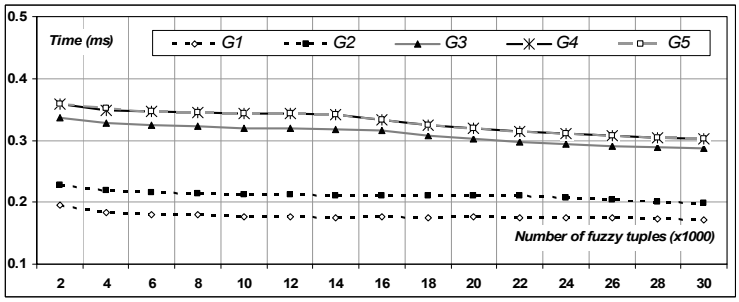
Figure 6. Algorithm scalability for 12.5% of data imprecision. (a) Actual time taken by our algorithm for partition trees G1-G5 (presented in Fig. 3). (b) Average time to traverse a single record for the same similarity relations.

in Figs. 5(b), 6(b), and 7(b)). This behavior is caused by almost constant data structures initialization costs (used mainly for loading a partition tree to the primary memory), which typically have a more significant influence on the algorithm average running time when considered in the context of smaller datasets. With each type of hierarchy, the average time of the defuzzification per tuple seems to stabilize to a constant as the number of tuples increases. Based on the plots presented in Figs. 5(b), 6(b), and 7(b), we are happy to report that the algorithm maintained the linear average time performance for all investigated partition trees. Exactly the same behaviors have been found when using datasets with 12.5% of data imprecision and the ones with 75%.

Now, let us consider two partition trees in Fig. 8. $G3$ and $G7$ have the same heights ($H = 4$), but differ in the distributions of branching factors in individual levels. The hierarchy $G3$ starts splitting its equivalence classes earlier (at high levels of the partition tree), whereas intensity of splitting for $G7$ increases at the bottom of the tree. Although both hierarchies have 32 atomic equivalence classes at the $\alpha = 1.0$ level of the tree, in the case of $G7$ they are merged



(a)



(b)

Figure 7. Algorithms scalability for 75% of data imprecision. (a) Actual time taken by our algorithm for partition trees G1-G5 (presented in Fig. 3). (b) Average time for the same similarity relations.

quickly at the lower part of the tree. In contrast, the other hierarchy ($G3$) possesses more equivalence classes closer to the top of the tree, resulting in the bigger number of nodes at intermediate levels (see Fig. 8(a)-(b)). The resulting runtime behaviors for these two hierarchies with respect to the degree of data imprecision are plotted in Fig. 8(c), and the results of our scalability tests are presented in Fig. 8(d). Our data defuzzification algorithm would always take more time in the case of partition tree $G3$ compared to $G7$ as the algorithm needs, on average, to traverse more nodes when searching the hierarchy $G3$. This is caused by the fact that trees with a lower branching factor close to their root are providing better chances for larger subtrees of the partition tree being pruned earlier, when we perform depth first search (DFS) using non-atomic values as search keys.

Figs. 8(c)-(d) show that our data defuzzification algorithm performs better with hierarchy $G7$ than $G3$. This suggests that the low branching factor near the root of the partition tree and comparatively higher branching factor at the bottom is the preferred choice when given a task of flattening partition trees for the purpose of attribute-oriented defuzzification.

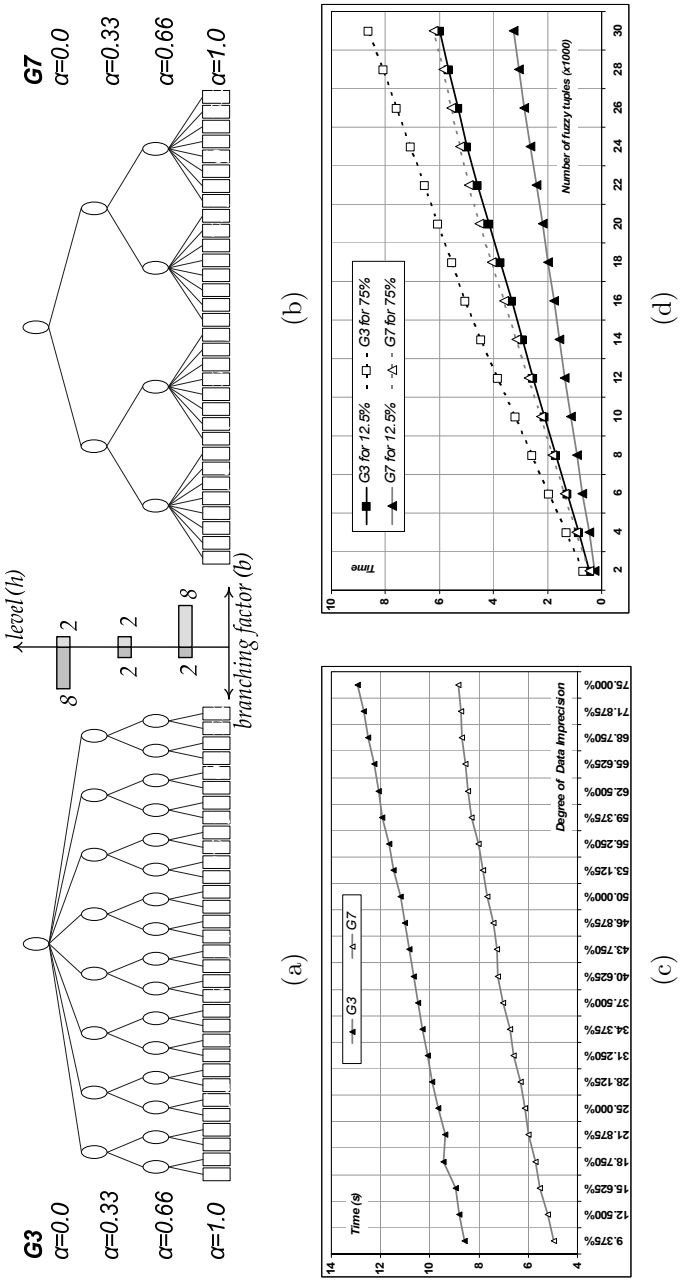


Figure 8. Impact of aggregating the equivalence classes at higher vs. lower levels of partition tree.

This is a very interesting behavior, as it is opposite to the strategies recommended for depth first search (DFS) using atomic search keys. In the case of atomic values, we want the trees to have bushy character at the top, as more expensive search at the top lets us prune the larger portions of a tree early, since only a single subtree is chosen at each level. In the case of sets of values (i.e. fuzzy data entries) used as search keys, a larger branching factor close to the root of the tree generates better chances of finding at least partial matches in the larger number of subtrees and therefore more nodes usually need to be checked (compare Figs. 9(b) and 9(c)). This generates additional computational costs and decreases performance of our *DFSearchForResemblances* procedure.

4. Conclusions

In this work we focused on the problem of interpretation of imprecise, categorical data entries in fuzzy relational databases. The paper shows a new, intuitive way in which a fuzzy tuple can be transformed to a collection of atomic values in a time-efficient manner. Our heuristic attribute-oriented defuzzification algorithm allows for transfer of fuzzy tuples to the form that allows analysis of such data via a majority of regular (i.e. non-fuzzy database specific) data mining algorithms.

By publishing this paper we hope to encourage scientists working in fuzzy databases area to popularize alternative fuzzy data interpretation algorithms allowing for analysis of imprecise data using well known classic data mining techniques. Our scalability tests for the developed algorithm are encouraging as they show that the fuzzy collections can be transformed to the atomic values efficiently. Our experiments show that time required to run our data defuzzification algorithm is linearly proportional to the degree of imprecision occurring in the fuzzy data table.

Acknowledgment

Rafal Angryk would like to express his gratitude to his former Ph.D. advisor – Dr. Frederick Petry, whose questions provided initial motivation for this work. The author also thanks Dr. Denbigh Starkey, and Mrs. Jeannette Radcliffe, who provided valuable comments on this writing. Some parts of the experimental results were generated using the code developed by Mr. Shahriar Hossain.

In addition, the author would like to thank the Montana NASA EPSCoR Grant Consortium for sponsoring a part of this research (Award No. M166-05-Z3184).

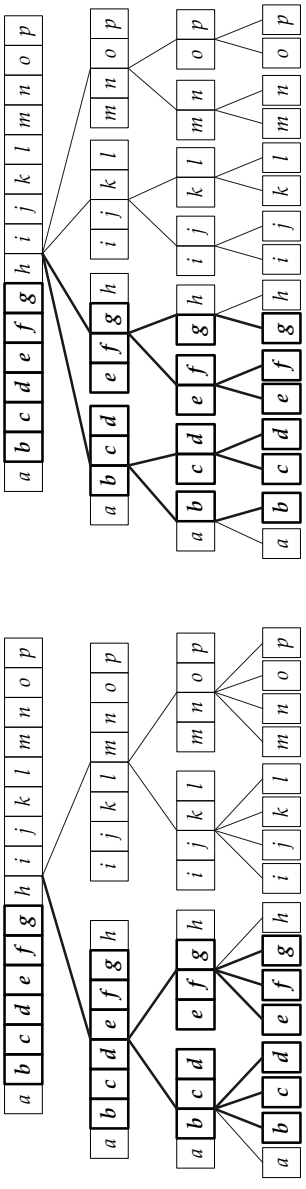


Figure 9. Comparison on search behavior for non-atomic search keys. (a) Non-atomic key-based DFS (for $Key=\{b,c,d,e,f,g\}$) in the tree with low branching factor (b=2) close to the root allows the benefits of pruning to occur earlier. (b) Non-atomic key-based DFS search in the tree with higher branching factor (b=4) close to the root.

References

Angryk, R. (2006) Similarity-driven Defuzzification of Fuzzy Tuples for Entropy-based Data Classification Purposes. *Proc. 15th IEEE Int. Conf. Fuzzy Systems (FUZZ-IEEE'06)*, Vancouver, Canada, July 2006, 1490-1498.

Angryk, R. and Petry, F. (2007) Discovery of generalized knowledge from Proximity-and Similarity-based Fuzzy Relational Databases. *International Journal of Intelligent Systems* **22** (7), 763-779.

Angryk, R. and Petry, F. (2005) Mining Multi-Level Associations with Fuzzy Hierarchies. *Proc. 14th IEEE Int. Conf. Fuzzy Systems (FUZZ-IEEE '05)*, Reno, NV, USA, May 2005, 785-790.

Baldwin, J.F. and Zhou, S.Q. (1984) Fuzzy relational inference language. *Fuzzy Sets and Systems* **14** (2), 155-174.

Buckles, B.P. and Petry, F.E. (1982) A fuzzy representation of data for relational databases. *Fuzzy Sets and Systems* **7** (3), 213-226.

Buckles, B.P. and Petry, F.E. (1983) Information-theoretic characterization of fuzzy relational databases. *IEEE Transactions on Systems, Man, and Cybernetics* **13** (1), 74-77.

Chuang, S.-L. and Chien, L.-F. (2002) Towards automatic generation of query taxonomy: a hierarchical query clustering approach. *Proc. 2nd IEEE Int. Conf. Data Mining (ICDM-IEEE '02)*, Maebashi City, Japan, December 2002, 75–82.

Chuang, S.-L. and Chien, L.-F. (2004) A practical Web-based approach to generating topic hierarchy for text segments. *Proc. of Conf. Information and Knowledge Management (CIKM'04)*, Washington, DC, November 2004, 127-136.

Codd, E.F. (1970) A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM* **13** (6), 377-387.

Han, J., Cai, Y. and Cercone, N. (1992) Knowledge discovery in databases: An attribute-oriented approach. *Proc. 18th Int. Conf. Very Large Data Bases (VLDB '92)*, Vancouver, Canada, 547-559.

Han, J., Cai, Y. and Cercone, N. (1993) Data-Driven Discovery of Quantitative Rules in Relational Databases. *IEEE Transactions on Knowledge and Data Engineering* **5** (1), 29-40.

Han, J. and Kamber, M. (2006) *Data Mining: Concepts and Techniques.* 2^{nd} Edition, Morgan Kaufmann, San Francisco, CA.

Kacprzyk, J. and Zadrozny, S. (2005) Linguistic database summaries and their protoforms: towards natural language based knowledge discovery tools. *Information Sciences* **173** (4), 281-304.

Kantardzic, M. and Zurada, J. (2005) *New Generation of Data Mining Applications.* IEEE Press and John Wiley.

Medina, J.M., Pons, O. and Vila, M.A. (1994) GEFRED: a generalized model of fuzzy relational databases. *Information Sciences—Informatics and Computer Science: An International Journal* **76** (1-2), 87-109.

Petry, F.E. (1996) *Fuzzy Databases: Principles and Applications*. Kluwer Academic Publishers, Boston, MA.

Prade, H. and Testemale, C. (1984) Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries. *Information Sciences* **34** (2), 115-143.

Raju, K.V.S.V.N. and Majumdar, A.K. (1988) Fuzzy functional dependencies and lossless join decomposition of fuzzy relational database systems. *ACM Transactions on Database Systems* **13** (2), 129-166.

Rundensteiner, E.A., Hawkes, L.W. and Bandler, W. (1989) On nearness measures in fuzzy relational data models. *International Journal of Approximate Reasoning* **3** (3), 267-298.

Shenoi, S. and Melton, A. (1989) Proximity Relations in the Fuzzy Relational Database Model. *International Journal of Fuzzy Sets and Systems* **31** (3), 285-296.

Tamura, S., Higuchi, S. and Tanaka, K. (1971) Pattern Classification Based on Fuzzy Relations. *IEEE Transactions on Systems, Man, and Cybernetics* **1** (1), 61-66.

Urrutia, A., Galindo, J., Jimenz, L. and Piattini, M. (2006) Data Modeling Dealing With Uncertainty in Fuzzy Logic. In: D. Avison, S. Elliot, J. Krogstie, J. Pries-Heje, eds., *The Past and Future of Information Systems: 1976–2006 and Beyond*. Series: *IFIP International Federation for Information Processing* **214**, Springer, Boston, 201-217.

Wall, B., Richter, N. and Angryk, R. (2005) Creating Concept Hierarchies in an Information Retrieval System. *Proc. 5th IEEE Int. Conf. Data Mining (ICDM-IEEE '05), Workshop on Foundations of Semantic Oriented Data and Web Mining*, Houston, TX, USA, November 2005, 99-105.

Weka 3 (2009) Data Mining Software in Java (March 17^{th}, 2009), http://www.cs.waikato.ac.nz/ml/weka/

Yager, R.R. and Petry, F.E. (2006) A Multicriteria Approach to Data Summarization Using Concept Ontologies. *IEEE Transactions on Systems, Man, and Cybernetics* **14** (6), 767-780.

Zadeh, L.A. (1970) Similarity relations and fuzzy orderings. *Information Sciences* **3** (2), 177-200.

Zemankova-Leech, M. and Kandel, A. (1984) *Fuzzy Relational Databases - a Key to Expert Systems*. Interdisciplinary Systems Research, Verlag TUV, Rheinland, Koln.