International Journal of Approximate Reasoning 51 (2010) 895–911

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

journal homepage: www.elsevier.com/locate/ijar

Heuristic algorithm for interpretation of multi-valued attributes in similarity-based fuzzy relational databases

Rafal A. Angryk [a,*], Jacek Czerniak [b]

[a] Department of Computer Science, Montana State University, Bozeman, MT 59717-3880, USA
[b] Systems Research Institute, Polish Academy of Sciences, Laboratory of Intelligent Systems, 01-447 Warszawa, Poland

ARTICLE INFO

Article history:
Available online 15 June 2010

Keywords:
Similarity-based fuzzy relational databases
Multi-valued entries
Taxonomic symbolic attributes
Fuzzy similarity relation
Data mining

ABSTRACT

In this work, we are presenting implementation details and extended scalability tests of the heuristic algorithm, which we had used in the past [1,2] to discover knowledge from multi-valued data entries stored in similarity-based fuzzy relational databases. The multi-valued symbolic descriptors, characterizing individual attributes of database records, are commonly used in similarity-based fuzzy databases to reflect uncertainty about the recorded observation. In this paper, we present an algorithm, which we developed to precisely interpret such non-atomic values and to transfer the fuzzy database tuples to the forms acceptable for many regular (i.e. atomic values based) data mining algorithms.

1. Introduction

In recent years, we have observed a very active growth of the field of data mining, both in research and real-life applications. In computer-related literature we can find multiple references to the term "data avalanche", widely used by Gray [3] as a description of the current Internet era. We have multiple web portals registering users' click-streams, clerks in the stores asking their customers to fill out surveys on the Web, and multiple types of electronic forms for gathering peoples' feedback, starting from medical research and finishing on well-known marketing explorations. The business world quickly discovered profits coming from the extended large-scale data analysis, and is aggressively trying to take advantage of the capabilities offered by the data mining.

At the same time, data-related research in fuzzy sets community seems to be split into two parallel tracks. There are people intensively working on improving fuzzy database models in the context of fuzzy data modeling and querying techniques, and a separate group of researchers developing fast fuzzy analysis techniques, entirely focused on processing regular (i.e. non-fuzzy) relational databases. By publishing this work we want to encourage research on merging these two paths. In our opinion, to guarantee the successful future of the fuzzy databases field, we need to develop techniques that quickly transfer fuzzy data into formats that are easy to deploy by the data miners working for large, business-oriented companies. This may encourage them to incorporate capabilities offered by current fuzzy database models into their own, large relational database systems. In this work we discuss a heuristic algorithm that allows transfer of fuzzy tuples (containing collections of values in each attribute) from tables in similarity-based fuzzy relational databases into sets of records containing only atomic values, which are compatible with Codd's First Normal Form (1NF) definition [4] and can be easily analyzed using regular data mining techniques [5].

In the pages that follow we first briefly review the capabilities of similarity-bases fuzzy relational database model, and then characterize the type of fuzzy data, which we will focus on, from data miners' perspective. We next discuss two

* Corresponding author.
E-mail addresses: angryk@cs.montana.edu (R.A. Angryk), jacek.czerniak@ibspan.waw.pl (J. Czerniak).

0888-613X/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ijar.2010.05.001

alternative approaches to the interpretation of fuzzy records. After this, we present our heuristic algorithm, followed by a theoretical investigation on its computational cost and extended experimental results on large, artificially created fuzzy data tables. In closing, we present a brief example of using our technique for imprecise data classification and suggest some directions for future research.

2. Related works

2.1. Similarity-based fuzzy relational database model

The similarity-based fuzzy model of a relational database, proposed originally by Buckles and Petry [6–8], is an extension of the ordinary relational database, developed by Codd [4]. The fuzzy model uses a binary fuzzy similarity relation, proposed by Zadeh in [9], which extends a classical equivalence relation. An interesting discussion of fuzzy relational database properties was recently presented by Kumar De et al. [10]. For readers deeply interested in the topic, we would also recommend works published by Parsons [11] and Ma [12,13].

There are two fundamental properties of the similarity-based fuzzy relational database, distinguishing it from a regular relational database model: (1) acceptance of non-atomic values when characterizing properties (i.e. attributes) of recorded entities, and (2) use of multi-level equivalence classes, derived from domain-specific fuzzy similarity relations applied in the place of traditional equivalence relations.

The first property can be explained formally as follows. If we denote a set of acceptable attribute values as A_j, and we let a_{ij} symbolize a jth attribute entry of the ith tuple. Instead of $a_{ij} \in A_j$ (expected in each regular relational database, in agreement with definition of the Codd's First Normal Form (1NF) [4]) the more general case of $a_{ij} \subseteq A_j$ is allowed in the fuzzy database. That is, any member of the power set of the attribute's domain values can be used as a valid attribute entry. Buckles and Petry [6] proposed that in the case when a particular entity's attribute cannot be clearly characterized by an atomic descriptor, such uncertainty can be reflected by insertion of multiple attribute values, representing fuzzy data entry.

The second feature, characterizing the similarity-based fuzzy databases, is the substitution of the ordinary equivalence relation with Zadeh's fuzzy similarity relation [9] of which the equivalence relation is a special case. Each of the attributes in the fuzzy database has its own domain-specific fuzzy similarity table, which contains the degrees of similarity between all values occurring for the particular attribute (see Table 1). Disjoint classes of attribute values, considered to be equivalent at a specific α-level, can be extracted from the similarity table (they are marked by shadings in Table 1). These groups of attribute values were referred by Zadeh [9] as *equivalence classes*. The existence of a fuzzy similarity relation for a particular attribute A_j of a fuzzy database table allows for the extraction of a concept hierarchy (called by Zadeh [9] a *partition tree*), representing disjoint equivalence classes on multiple α levels (see Fig. 1). From the propagation of shadings in Table 1, we can observe that the equivalence classes are clearly separated and have a nested character [2,9].

For the purpose of this work, we will introduce a formal notation that we are going to use when referring to partition trees. A partition tree, defined for a single attribute in a fuzzy database model, can be interpreted as a set of interconnected

Table 1
Fuzzy similarity table for domain COUNTRY [2].

	Canada	USA	Mexico	Colombia	Venezuela	Australia	New Zealand
Canada	*1.0*	*0.8*	*0.8*	*0.4*	*0.4*	*0.0*	*0.0*
USA	*0.8*	*1.0*	*0.8*	*0.4*	*0.4*	*0.0*	*0.0*
Mexico	*0.8*	*0.8*	*1.0*	*0.4*	*0.4*	*0.0*	*0.0*
Colombia	*0.4*	*0.4*	*0.4*	*1.0*	*0.8*	*0.0*	*0.0*
Venezuela	*0.4*	*0.4*	*0.4*	*0.8*	*1.0*	*0.0*	*0.0*
Australia	*0.0*	*0.0*	*0.0*	*0.0*	*0.0*	*1.0*	*0.8*
New Zealand	*0.0*	*0.0*	*0.0*	*0.0*	*0.0*	*0.8*	*1.0*

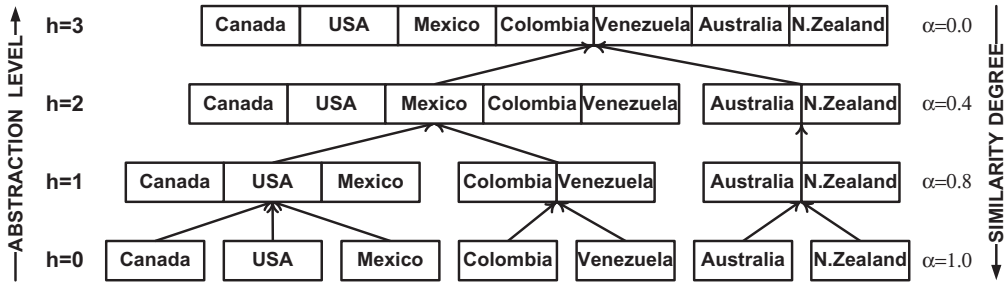

Fig. 1. Partition tree for domain COUNTRY, built on the basis of Table 1. The increase of conceptual abstraction (denoted by h) in the partition tree is reflected by decreasing degrees of fuzzy similarity α; lack of data abstraction ($h = 0$) complies with the 1-cut of the similarity relation ($\alpha = 1.0$).

ordered pairs (i.e. abstraction levels) denoted as G, such that $G = \{(A^h, \alpha_h): 0 \leqslant h < H, \alpha_h \in [0.0, 1.0]\}$, where $\|A^h\| > \|A^{h+1}\|$ (the granularity increase property is a natural consequence of the self-nested distribution of equivalence classes in the neighboring levels of the partition tree), and $\alpha_h > \alpha_{h+1}$ (caused by the abstraction increase when we travel from the bottom to the top of the tree). In our notation, $A^h = \left\{a_1^h, a_2^h, \ldots, a_{\|A^h\|}^h\right\}$ is the set of equivalence classes defined for hth – abstraction level of the tree, where a_m^h denotes the mth node (i.e. equivalence class) at the hth level of the partition tree. For instance, a set of equivalence classes at 0.8-level in Fig. 1 (denoted as set of A^1) consists of three nodes (i.e. $\|A^1\| = 3$): $\{a_1^1 = \{Canada, USA, Mexico\}, a_2^1 = \{Colombia, Venezuela\}, a_3^1 = \{Australia, N.\ Zealand\}\}$. The nested inter-level (between the different levels) and disjoint intra-level (within the individual levels) character of nodes (i.e. equivalence classes) in the partition trees is assumed in the remaining parts of this work.

2.2. Analysis of multi-valued entries in taxonomic categorical attributes

In recently published book [14], edited by Bock and Diday, we can find a comprehensive collection of the latest works related to the extraction of statistical information from the non-numeric data. Based on the classification of data types presented in the book, the data considered in this paper can be classified as *taxonomic symbolic variables*. These types of entries are expected to carry no quantitative meaning, but yet the values can be nonlinearly ordered in a form of rooted, hierarchical tree, called in this branch of the literature [14,15] a taxonomy. Zadeh's partition tree [9], as presented in the Fig. 1, is a good example of such taxonomy. Our fuzzy data entry, in this stream of the literature [14,15], is simply called a multi-valued variable. While multi-valued descriptors are frequently used (e.g. in online surveys, medical forms), the problem of their analysis, however, still remains a challenge [14,15].

In [14], chapter six has been devoted almost entirely to the problem of derivation of basic description statistics (in particular: medians, modes and histograms) from multi-valued symbolic data. The authors propose the transformation of non-atomic values to collection of pairs in the format of (a, μ), where a stands for a singleton (i.e. atomic) symbolic value, and μ represents a's observed frequency. This is achieved by extending the classical definition of frequency distribution. For a multi-valued variable, the new definition states [14,15] that the number reflecting the observed frequency (i.e. count of a's appearances in the data set) can be a positive real, instead of a positive integer, as is the case for each single-valued variable.

In [14], the authors mention that other definitions of frequency distributions can be proposed, suggesting taking into account the natural dependencies between the symbolic values, as reflected by the provided attribute's taxonomy. Since Zadeh's partition trees [9] used in fuzzy databases can be interpreted as attribute taxonomies, we defined [2] the frequency distribution for a fuzzy data entry, denoted in Section 2.1 by a_{ij} (such that $a_{ij} \subseteq A_j^0$), as the list of all atomic descriptors in the finite domain of A_j^h, together with the percentage of instances of each of the values in this data point. This lets us specialize each uncertain (i.e. multi-valued) attribute entry a_{ij} into a sequence of pairs (a_m^h, μ_{ij}), where a_m^h denotes the atomic attribute value (i.e. a single equivalence class) at the preferred hth level of partition tree (i.e. h is constant for all pairs), and μ_{ij} represents a fraction of the imprecise record the atomic a_m^h attribute value represents. To make sure each record in our fuzzy database has equal weight and all record's attributes are treated evenly during multi-attribute specialization process, we want our record's fractions to maintain the following condition: $\sum_{a_m^h \in a_{ij}} \mu_{ij} = 1.0$.

3. Interpretation of imprecise tuples

Since the fuzzy database model permits the reflection of uncertainty about the registered fact via insertion of multiple attribute descriptors, to let us use regular (i.e. non-fuzzy) data mining techniques to analyze the imprecise data [1,2,16] we needed to develop a mechanism allowing consistent interpretation of the multi-valued data. In other words, we wanted to map a collection of non-atomic values stored in a fuzzy tuple into a set of atomic descriptors (i.e. singletons), that are compatible with 1NF definition and can be analyzed using regular data mining algorithms (i.e. defuzzify our fuzzy database table).

Having background knowledge available in form of fuzzy similarity relations, we were naturally interested in making sure that the interpretation of imprecise attribute values is linked to the fuzzy similarity relation defined for this attribute (e.g. Table 1). The hierarchical distribution of equivalence classes in the partition tree (e.g. Fig. 1) can provide data miners with useful and important knowledge that needs to be utilized in data mining applications, and should allow for consistent and accurate interpretation of imprecise data. Different attribute values can be considered as identical or totally dissimilar, depending on the preferred level of generalization hierarchy. Fuzzy similarity relations give us the ability to analyze the same data on multiple levels of granularity. Moreover, knowing that the partition tree reflects actual similarities between data values, analysis of entered values may provide data miners with a hint on the granularity level (i.e. α) the persons, who were entering values, were able (or willing) to recognize and register.

In the next section, we will discuss two major approaches to the interpretation of fuzzy tuples [2]: (1) *lossless approach*, preserving all original entries stored in the fuzzy database table, and (2) *lossy approach*, which can transfer the original data entries into one common level of abstraction in our partition tree. In the remaining sections, we will show an intuitive example of lossy approach to interpretation of fuzzy records and introduce details of our algorithm. This will be followed by theoretical and experimental evaluations of our algorithm.

3.1. Lossless and lossy approaches to fuzzy tuples interpretation

Lossless approach to fuzzy tuples' interpretations is the most universal, and at the same time the most extreme case, as it separates every combination of values (i.e. any subset of domain values actually entered for a particular attribute of a registered entity) and maps it to a unique (and therefore – atomic) descriptor. Since the mapping is one-to-one, we call it *lossless*, as originally entered values can be backtracked if necessary (assuming the one-to-one mapping has been kept record of). For instance, {*Canada*} would be interpreted as {*Canada*}, while entry {*Canada, USA*} would be interpreted as some type of unique combination of these two values, i.e. {*CanadaUSA*}. This process can generate n unique combinations of descriptors for a single domain, such that: $n \leqslant 2^{\|A^0\|}$. We can expect $n = \|A^0\|$ for an attribute that contains only precise data, and $n = \sum_{h=0}^{H-1} \|A^h\|$ for an imprecise attribute with a fuzzy similarity relation that reflects perfectly all dependencies occurring in real-life data (i.e. every imprecise record is a reflection of one of equivalence classes already specified in the appropriate partition tree). However, since a database relation is a subset of the cross product of all domains' values, we may end up with $2^{\|A_1^0\|} \times 2^{\|A_2^0\|} \times \cdots \times 2^{\|A_m^0\|}$ unique descriptors for a fuzzy database table containing m attributes.

Despite some advantages of the *lossless* interpretations (accuracy of original data representation, preservation of original data size and character, backtracking of the original entries, etc.), there are many good reasons to avoid this type of data interpretation in practice. As we showed above, the *lossless* interpretation may cause exponential growth of new "atomic" descriptors, while maintaining the original size of the data table. This can have a significant influence on results generated by association rules and classification algorithms, as it causes considerable expansion of granularity of our data space in dimensions which have a highly imprecise character. First, support for some individual association rules may be decreased, making them less important for decision makers. Second, many of well known classification algorithms are strongly biased toward attributes with larger number of descriptors, interpreting them as attributes allowing for more pure separation of classes [17–19]. Finally, this type of interpretation can be also very confusing for a customer, who may have trouble interpreting the artificially created one-to-one mapping (e.g. {*CanadaUSA*} vs. {*USA*}).

The *lossy approach* to the interpretation of fuzzy tuples is driven by a need for the transformation of all originally entered values, representing different degrees of imprecision, to a single, common abstraction level, where atomic interpretations can be assigned to them. Such a transformation is going to generate the $\|A^\ell\|$ of unique descriptors for a single domain, such that $\| A^0\| \leqslant \|A^\ell\| \leqslant \|A^{H-1}\|$, where ℓ symbolizes the preferred abstraction level and $\|A^\ell\|$ denotes the number of unique descriptors (i.e. equivalence classes) at that level. To quickly explain our concept of *lossy* interpretation, we will use a trivial example. If we have two values entered in the column COUNTRY_OF_ORIGIN: {*Canada, USA*}, we could interpret such a record as two halves of a record – one having an atomic value {*Canada*}, and another saying precisely {*USA*}, each with half of the vote carried by the original imprecise/fuzzy record (which has a total a value of unity). The transformation of fuzzy data tables to a certain granularity level may permit the reduction of the size of the original data repository (leading in consequence to faster data processing) and provide a more human-friendly data representation.

The most extreme case of lossy interpretation would be the transformation of a fuzzy database relation into a defuzzified table, where all entries are atomic and located at the lowest level of the partition tree ($\ell = 0$). We will call this case a *complete data specialization*, as it requires transformation of all imprecise (i.e. multi-valued) entries into the equivalence classes distinguished at the very bottom of the partition tree (where $\alpha = 1.0$ and $h = 0$).

As mentioned at the end of Section 2.2, this may require development of a technique allowing appropriate splitting of a vote, carried by an individual fuzzy database tuple, into multiple parts, reflecting the number of originally entered non-atomic descriptors and similarities between them (i.e. our imprecision). One of the major contributions of this paper is the introduction of one possible heuristic that incorporates a fuzzy similarity relation into the *lossy specialization* process, which is going to be discussed in detail shortly.

Obviously, with the *lossy* interpretation, we are not obligated to always choose only the terms at the lowest level of the partition tree as our data representation. We can choose any (e.g. customer-preferred) level of data granularity and transfer the whole data set to the level of their preference. Such an approach would require both (1) a generalization of values, representing equivalence classes located at the levels of granularity lower than the preferred level; as well as (2) the specialization of data entries reflecting equivalence classes at the levels of abstraction higher than the one desired by our customer. The last case can be interpreted as a form of *partial data specialization* (i.e. a specialization of data, which does not require reaching to a bottom of partition tree, but stops at the preferred α-level). For instance, if we would prefer to analyze our data based on the CONTINENT-level (represented by 0.8-level in Table 1), three sets of values (originally entered into COUNTRY_OF_ORIGIN attribute of three separate fuzzy database records), could be transformed to the following forms: (1) {*Canada*, *USA*, *Mexico*} → {*N. America*}, (2) {*Canada, USA*} → {*N. America*}, (3) {*Canada, Australia*} → 1/2 as {*N. America*}, and 1/2 as {*Australia*}.

Despite our focus on higher (i.e. 0.8) abstraction level, the last entry still generated a split of the record's vote, as the descriptors *Canada* and *Australia* cannot be treated as identical at the CONTINENT-level. It shows that in the case of the last tuple some information about data imprecision has been maintained, despite of moving our interpretation to more abstract level.

As data generalization has been extensively investigated in the past ([16,20–23]) and *lossless* data interpretation is very ineffective in practice, in this work we focus our attention almost entirely on *complete lossy data specialization* (i.e. reaching with our lossy specialization to the 0-abstraction level). We have chosen to do so to simplify our presentation, knowing that completely defuzzified data can be always generalized to a more abstract level. Due to the disjoint and nested character of

equivalence classes in Zedeh's partition trees, *data-generalization* of fuzzy tuples has a typically straightforward character [22–25]. We can simply replace the originally inserted values with more abstract concepts (i.e. equivalence classes occurring at the higher level of partition tree), where the original values belong. Then we can treat a record with originally lower-level descriptors as a member of higher-level equivalence class. If the similarity table accurately reflects real-life dependencies among the data, the vote of the generalized tuple may never need to be split, as the equivalence classes in the partition hierarchy have a nested character.

When performing *lossy* specializations of fuzzy tuples, we have to extend the output relation with the additional attribute COUNT. The attribute is necessary to preserve a consistent representation of the original data after the *lossy* transformation, as this transformation may cause merging identical fractions of originally separate records (e.g. half of the vote with now atomic value {*N. America*} can be merged with other records that contain a descriptor belonging to the same equivalence class, as long as values in all remaining attributes are the same). In such case, an accurate count of votes is essential to guarantee proper data mining results. We want to protect every original tuple from being counted more or less than once when the *lossy* data interpretations are performed. By making sure that we represent each record of the original fuzzy database table as a single vote, when having it either split (i.e. fractioned) into multiple equivalence subclasses (what may occur when we decide to transfer a highly imprecise tuple to match equivalence classes at the lower level of partition tree), or generalized (i.e. summarized) to a more abstract equivalence class(es) (when we are going up in the partition tree), we are guaranteed that original proportions among the data points remain accurate in our defuzzified output data table.

After performing *complete data specialization* (i.e. a specialization reaching the bottom of the partition tree) on the large fuzzy database table, we are capable of reducing the size of the original relation by merging all fractions of tuples that look identical after the transformation (from possibly $2^{\|A^0\|}$ attribute values we are back to $\|A^0\|$ again). The additional attribute COUNT, representing the propagation of original votes, lets us maintain some information about the distribution of the originally entered values. Unfortunately, an exact image of uncertainty distribution (i.e. assignment of non-atomic values to specific tuples) cannot be maintained anymore. That is the reason we called this approach a *lossy* one.

3.2. Similarity-driven complete lossy specialization of fuzzy tuples – an example

In this section we discuss a *complete data specialization*, as the most extreme case of lossy interpretation. Our approach is based on partial vote propagation, where a single vote, corresponding to one fuzzy database tuple, is partitioned before being assigned to the concepts separated in the lower levels of the partition hierarchy. To quickly explain our main idea, we will start with a simple example. Let us ask ourselves in what COUNTRY ($\alpha = 1.0$ level in Fig. 1) should we expect to find a drugs dealer who, as a not-confirmed report says, was recently seen in {*Canada, Colombia, Venezuela*}?

During *complete data specialization* we want the fractions of an imprecise vote to be assigned to separate concepts from A^0 set in such way that these fractions reflect each of the originally inserted attribute values, as well as the similarities between these attribute values. The most trivial solution would be to split the vote equally among all inserted descriptors: {*Canada*|*0.333, Colombia*|*0.333, Venezuela*|*0.333*}. This approach, called *Averaging*, however does not take into consideration important real-life dependencies, which are reflected not only in the number of inserted descriptors, but also in their similarity (reflected in our partition tree).

We propose here a replacement of the even distribution of a vote with a nonlinear spread, dependent both on the similarity of inserted values and on their quantity. 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. We call them *subsets of resemblances* (e.g. {*Colombia, Venezuela*} from the above example). Then we use them as a basis for calculating a more intuitive distribution of a vote's fractions. An important aspect of our approach is the extraction of the *subsets of resemblances* 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. Repetitive extraction of such subsets could unbalance the original dependencies among inserted values, skewing significantly the results of our interpretation toward *subsets of resemblances* occurring at the very low levels of partition trees.

Our algorithm is quite straightforward and we will discuss it in detail in the next section. For now, assume that you are given (1) a set of attribute values inserted as a description of particular entity (i.e. our imprecise attribute entry, e.g. {*Canada, Colombia, Venezuela*}), and (2) a tree-like structure reflecting a partition tree for the particular attribute (e.g. Fig. 2). Using

Table 2
Subsets of resemblances for the analyzed example.

Subsets of resemblances and their similarity levels (r_p,α_p)	*COMMENTS*
{*Canada, Colombia, Venezuela*}|*0.0*	*STORED*
{*Canada, Colombia, Venezuela*}|*0.4*	*UPDATED*
{*Canada*}|*0.8*	*STORED*
{*Canada*}|*1.0*	*UPDATED*
{*Colombia, Venezuela*}|*0.8*	*STORED*
{*Colombia*}|*1.0*	*STORED*
{*Venezuela*}|*1.0*	*STORED*

these inputs, we want to extract a table, which includes (a) the list of all subsets of resemblances occurring in the given fuzzy tuple, and (b) the highest degree of similarity (α) of their common occurrence (see Table 2 for an example). Then, we can use the list to intuitively distribute fractions of the original record (e.g. Fig.3), where by "intuitive distribution" we understand making sure that the sum of all vote's fractions generates unity and that the size of individual fractions reflects distribution of relevant equivalence classes in our fuzzy similarity relation.

Our algorithm uses preorder recursive traversal for searching a partition tree. The partition tree is searched starting from its root and, if any subset of the given set of descriptors 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 subtree of the partition tree.

Before doing a formal presentation of our algorithm, we will present an example of such a search for subsets of resemblances in a tuple with the earlier mentioned values {*Canada, Colombia, Venezuela*}. Behavior of our algorithm is depicted in Fig. 2. Numbers on the links in the tree represent the order in which the particular *subsets of resemblances* were discovered. After extracting the *subsets of resemblances*, we apply a simple summarization of α values as a measure reflecting both the frequency of occurrence of the particular attribute values in the subsets of similarities, as well as the abstraction level of these occurrences. Since the country *Canada* was found only twice in the *subsets of resemblances*, we assigned it a grade 1.4 (i.e. 1.0 + 0.4). The remaining attribute values were graded as follows: *Colombia*|(1.0 + 0.8 + 0.4) = *Colombia*|2.2, *Venezuela*|(1.0 + 0.8 + 0.4) = *Venezuela*|2.2.

In the next step, we add all generated grades (1.4 + 2.2 + 2.2 = 5.8) to normalize grades finally assigned to each of the participating attribute values: *Canada*|(1.4/5.8) = *Canada*|0.24, *Colombia*|(2.2/5.8) = *Colombia*|0.38, *Venezuela*|(2.2/5.8) = *Venezuela*|0.38.

This leads to the new distribution of the vote's fractions, which more accurately reflects real-life dependencies than a linear weighting approach. We called this set, which is the final output of our algorithm and consists of pairs denoted by $\left(a_m^h, \mu_{ij}\right)$ in Section 2.2 (where $h = 0$, for *complete specialization* process), a Specialized Tuple and denoted it with the letter *F* in Table 3. Normalization of the initial grades guarantees that each of the records is represented as unity, despite being variously distributed at the desired specialization level. In the case of *partial data-specialization*, requiring stopping interpretation at a higher abstraction level, the depth-first search (DFS) through our partition tree needs to be simply stopped at the preferred level of abstraction.

3.3. Detailed presentation of algorithm for similarity-driven complete lossy specialization

Now, we will discuss details of our data specialization algorithm, which is presented in Table 3. As the input, we assume: (1) a multi-valued data entry to one attribute of a single fuzzy tuple, which we refer to as an *Imprecise Attribute Entry*, de-

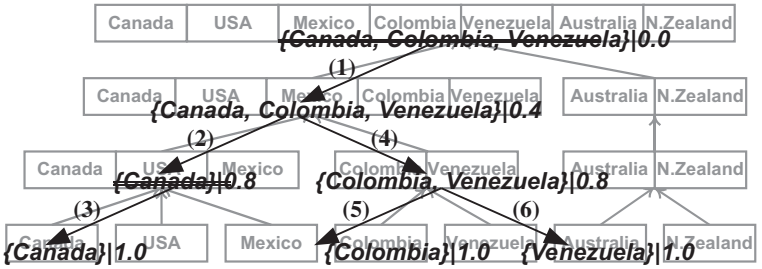

Fig. 2. Subsets of resemblances extracted from the partition tree in Fig. 1. (See Table 2 for results)

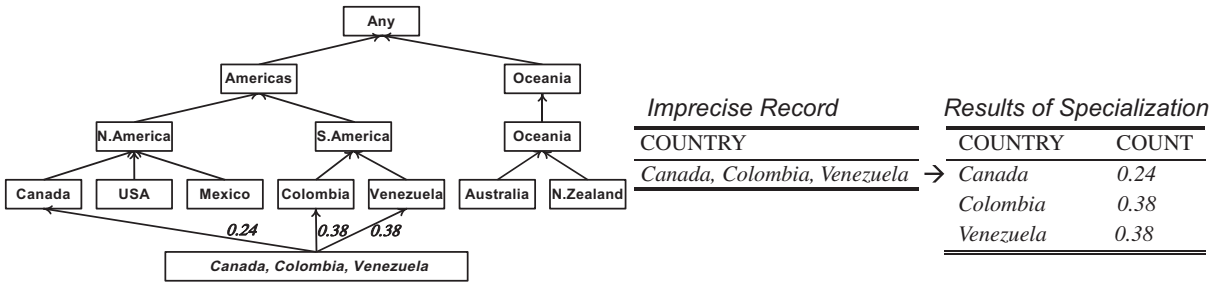

Fig. 3. Similarity-based vote distribution for the analyzed record with uncertainty (on the left), and the results of complete lossy specialization of that record (on the right).

noted in Table 3 as *T*, and (2) a *Partition Tree*, reflecting a fuzzy similarity relation specified for the given attribute, denoted as *G*. Our algorithm can be run successively on all attributes of a fuzzy database table that contain an imprecision (i.e. non-atomic values). To keep things simple, we decided to present our approach in a context of a single attribute.

On the output, we will get a collection of pairs in the format (*atomic attribute value, record's fraction*), denoted as (a_k^0, μ_k) in Table 3 (see Fig. 3 for an example), which reflects the complete lossy specialization of an individual fuzzy record with a single attribute. We denote the collection as a set *F*, and call it a *Specialized Tuple*. Note an important $\sum_{k=1}^{\|A^0\|}\|\mu_k = 1.0$ condition specified at the output section in Table 3; it assures that each fuzzy tuple maintains its original representation of individual database record's vote at the end of the data specialization process. The normalization of the imprecise record's fractions has been implemented in the lines 5–10 of the algorithm presented in Table 3. The normalization is necessary to keep the original proportions of information for data mining algorithms that follow our fuzzy data specialization process. By assigning each (imprecise or not) database record exactly one vote, we are making sure that the original proportions among multiple data points in our fuzzy database table are maintained through the whole data mining process.

The temporary structure, called the *Subset of Resemblances*, which was described in the previous section, is also specified in the output section of Table 3 and denoted as *R*.

In lines 1 and 2 in Table 3, we simply clear both output structures (*R* and *F*). In line 3 we generate all possible *Candidates for Subsets of Resemblances* (each non-empty subset of *T*, which is a candidate, is denoted in Table 3 by c_i), which are derived from the imprecise attribute entry *T* provided as the algorithm's input. It is important to order the candidates from the longest ($\|c_1\| = \|T\|$), to 1-element long structures ($\|c_{2^{\|T\|}-1}\| = 1$); this will allow us to reduce number of comparisons during the future depth-first search (DFS) through our partition tree, referred as *DFSearchForResemblances* in Table 3 (code lines 12–17). The preorder recursive traversal of the partition tree *G* is initiated in line 4; in our experiments we implemented *DFSearchForResemblances* as a recursive procedure.

In line 12, we check if we reached the bottom of our partition tree, or whether there are no more candidates for subsets of resemblances to be considered (i.e. our set *C* is empty). If at least one of these conditions is true, we complete the recursive DFS call in the given branch of *G* immediately (and return our search to its parent, which then can move to its next child/subtree). If this is not the case, we check if the first (i.e. currently largest) of the candidates (still available for the given subtree of *G*) occurs in the analyzed node a_m^h (i.e. in the currently searched equivalence class) of the partition tree *G*. If this is the case (i.e. c_1 is a subset of the node a_m^h), we send the candidate to update our subset of resemblances *R* (line 13). The update takes place in lines 18–21 (*UpdatePairOfResemblances* procedure). If this case does not occur (i.e. no match between a can-

Table 3
Algorithm for similarity-driven complete data specialization of multi-valued symbolic data entries.

INPUT: **Imprecise Attribute Entry**, denoted as *T*, such that $T \subseteq A^0$, where $2 < \|T\| \leqslant \|A^0\|$
Partition Tree, a set of ordered pairs denoted as *G*, such that $G = \{(A^h,\alpha_h): 0 \leqslant h < H, \alpha_h \in [0.0,1.0]\}$
$A^h = \left\{a_1^h, a_2^h, \ldots, a_{\|A^h\|}^h\right\}$ is the set of similarity classes defined for *h*th abstraction level, and a_m^h denotes the *m*th node (i.e. similarity class) in the *h*th level of *G*

OUTPUT: **Specialized Tuple**, a set of pairs denoted as *F*, such that $F = \{(a_k^0, \mu_k) : a_k^0 \in T, \mu_k \in [0.0, 1.0]\}$, where $\sum_{k=1}^{\|A^0\|}\mu_k = 1.0$
Subset of Resemblances, a set of ordered pairs denoted as *R*, such that $R = \{(r_p, \alpha_p) : r_p \subseteq a_m^h, (a_m^h, \alpha_p) \in G\}$, where $\alpha_p = MAX\{\alpha_h : (a_m^h, \alpha_h) \in G, r_p \subseteq a_m^h\}$

METHOD:

(1) $R = \{\emptyset\}$;
(2) $F = \{\emptyset\}$;
(3) Using similarity classes from A^0, such that $a_k^0 \in T$, generate a set of **Candidates for Subsets of Resemblances**, denoted as *C*, of non-empty subsets of *T* (ordered from the longest subset to the shortest one): $C = \{c_i: c_i \subseteq T, 1 \leqslant i \leqslant 2^{\|T\|} - 1, \|c_i\| \geqslant \|c_{i+1}\|\}$
(4) **for** (**each** similarity class a_j^{H-1} from the top of *G*) DFSearchForResemblances (a_j^{H-1}, C);
(5) **for** (**each** of the last $\|T\|$ elements of *C*, denoted as c_i)
(6) $\{\mu = 0.0$;
(7) **for** (**each** pair (r_p,α_p) in *R*) **if** $(c_i \subseteq r_p)$ $\mu = \mu + \alpha_p$;
(8) $F = F \cup \{(c_i,\mu)\};\}$
(9) $sum = \sum_{k=1}^{\|F\|}\mu_k$;
(10) **for** (**each** pair (a_k^0, μ_k) in *F*) $\mu_k = \frac{\mu_k}{sum}$;
(11) **return**;

Procedure DFSearchForResemblances (Node of Partition(Sub-) Tree a_m^h, Set of Candidates For Resemblances *C*)
(12) **if** $(h < 0 \vee C == \{\emptyset\})$ **return**;
(13) **if** $(c_1 \subseteq a_m^h)$ UpdatePairOfResemblances (c_1,α_h);
(14) **else** $\{C = C\backslash\{c_1\}$;
(15) DFSearchForResemblances $(a_m^h, C);\}$
(16) **for** (**each** a_j^{h-1}, s.t. $a_j^{h-1} \in DirectDescendentsOf(a_m^h)$) DFSearchForResemblances $\left(a_j^{h-1}, C\right)$;
(17) **return**;

Procedure UpdatePairOfResemblances (Candidate For Resemblances *c*, Similarity Level α)
(18) **for** (**each** pair (r_p,α_p) in *R*) **if** $(r_p == c)$ $\{\alpha_p = \alpha$;
(19) **return**;$\}$
(20) $R = R \cup \{(c,\alpha)\}$;
(21) **return**;

didate and a currently traversed node of the partition tree has been found), we continue to line 14, where we remove the unsuccessful candidate from the list of candidates that are still to be considered for this particular node of our partition tree and (later) its whole subtree. Then the search (with the temporally reduced number of candidates for subsets of resemblances: $C = C \backslash \{c_1\}$) continues in line 15. Here is how the local reduction of candidates for subsets of resemblances works. If the particular candidate for subset of resemblances, denoted by c_i, does not appear in the node of the partition tree, due to the self-nested character of our equivalence classes there is no reason to check if the same candidate appears in the subtree of that equivalence class – so we remove it from the local list of candidates C that need to be compared when traversing the related subtree. All subsets of that (removed) candidate are somewhere further in the candidates' line (note, that in line 3 we ordered the set C from the longest element, to the shortest), so we do not have to worry about omitting some candidates as we remove the irrelevant ones from the front of our candidates' list. The removal takes place in line 14. This is a very important step; since it allows reducing the number of necessary comparisons as we traverse down the partition tree (we start at the top, by comparing the largest equivalence class against our first, and the longest, candidate, which has the length of $||T||$, and gradually remove candidates from the set C, until it contains only a few candidates that have the length of 1, at the end of the traversed branch), and also lets us prune irrelevant branches of the tree G early (e.g. (1) if we already found matches between all the candidates no other equivalence classes in G need to be checked, and (2) if we empted our list of candidates when testing a particular node, its whole subtree can be cut off from the further search), what causes our algorithm to run faster.

After a candidate has been found within the currently searched equivalence class (i.e. test $c_1 \subseteq a_m^h$ in line 13 returned true), we move toward searching a subtree of this node (line 16), starting our comparisons of candidates from the most recent match (i.e. we do not have to compare the whole, original set C in each node).

The procedure *DFSearchForResemblances* is written in such a manner that it (1) avoids (i.e. cuts off) unrelated branches while traversing the tree (when no candidate c_i is matching the current node a_m^h, there is no sense to traverse the node's subtree), and (2) reduces the number of necessary comparisons between the candidates for subsets of resemblances (stored in local copy of C), and each of the nodes in the partition tree G (by removing from the current copy of C those candidates which have no chance for finding their matches in the given subtree of G). For example see Fig. 2; if the longest candidate for resemblances c_1 consists of {*Canada, Colombia, Venezuela*} (and therefore the whole set of candidates for resemblances must be: $C = \{$*{Canada, Colombia, Venezuela}, {Canada, Colombia}, {Canada, Venezuela}, {Colombia, Venezuela}, {Canada}, {Colombia}, {Venezuela}*$\}$), then our traversal would follow the path depicted in Fig. 2, where the order of our DFS traversal is marked by numbers 1–6. The local copy of C is systematically reduced as our *DFSearchForResemblances* procedure reaches smaller equivalence classes when traversing down the partition tree. E.g. the local copy of C is reduced to the last three 1-element long subsets, i.e. {*{Canada}, {Colombia}, {Venezuela}*}, after the branch with number 2 is taken (the largest element of the local set of C that matches local subtree is written on the top of each subtree in Fig. 2). From the example it becomes evident that the time complexity of the *DFSearchForResemblances* procedure is highly dependent on the dispersion of values inserted into our imprecise attribute entry T and their correlation with the equivalence classes in our partition tree G. As shown in Fig. 2, the unnecessary branches are always omitted by the *DFSearchForResemblances* procedure.

3.4. Theoretical investigation on time complexity of the algorithm

When trying to solve the problem of fuzzy tuples interpretation, two issues were cause for concern. The first (and the most important to us) was to develop an algorithm that follows our intuitive expectations and would allow for distribution of split record to reflect similarities between the entered values. The second was to develop a technique that we could successfully apply to interpret even large data sets. In this section we address the issue of time complexity. The evaluation is conducted in two steps. First, we will discuss time costs involved with interpretation of a single imprecise record (as given on the input in Table 3), and then we scale up our investigation to the case when a large database of imprecise records needs to be processed.

Our algorithm is built on a well-known "depth-first search (DFS) with cutoff" paradigm, and its time complexity is strongly dependent on the following factors: (1) the level of imprecision occurring in the fuzzy data table (i.e. the number of non-atomic values in imprecise tuples and the frequency of these imprecise records in the whole database table), and (2) the structure of the partition tree (often characterized by the height of the tree, and its average branching factor), reflecting fuzzy similarity relations, and (3) how well the multiple values, entered into the imprecise tuple, fit into the distribution of equivalence classes in the partition tree (referred in the literature [36,37] as "the probability parameter that characterizes the search"). The first factor is usually the largest, but since we want to investigate our approach in detail, we will start with the costs involved in the processing of a single imprecise record and searching through the partition tree. To do so, we need to focus on the 2nd and 3rd of the above factors.

It has been reported by researchers from IBM [37] that if H is the height of the tree (this is to be compatible with the notation presented in Fig. 1), and 2 is its average branching factor (this particular work concentrates on binary trees), "the average number of nodes visited by the [DFS] algorithm is as low as $O(H)$, and as high as $O(2^H)$ depending only on the value of the probability parameter that characterizes the search".

Let us investigate the worst of these two search scenarios first. To generalize our calculations we assume that (1) the average branching factor of the partition tree is b (not just 2), and (2) the total number of abstraction levels in the tree is H. If, following Zadeh's drawings of partition trees [9], we let the top of the tree to have only 1 node (i.e. one large equivalence

class that describes all values of A_j), then the next level has b nodes, the level after that has b^2 nodes,..., and the last level (where $\alpha = 1.0$) has b^{H-1} vertices. Hence, if we happen to have a data record that is so imprecise that it includes in its attribute *all* of the domain values (i.e.$\|T\| = \|A^0\|$), the *DFSearchForResemblances* procedure would need to search through all the vertices in the tree, and the total number of vertices is: $V = \sum_{h=0}^{H-1} b^h$, where h is the counter for the levels of the partition tree. The right-hand side of the last equation (for $b \neq 1$) represents a geometric series, which allows us to transfer the whole equation to the form: $V = \frac{b^H-1}{b-1}$, which is asymptotic to the function $O(b^H)$. Although the cost may initially look alarming (due to its exponential growth), we need to realize that the b^H number simply represents the total number of nodes in our partition tree ($O(b^H) = O(V)$), and it is constant for each of the database attributes. The number only characterizes the worst-case search scenario, when processing of the entire tree is needed, which can be caused *only* by a database record which contains *all possible values* from the original attribute's domain. It is important to realize that such an extreme case occurs very rarely, as these types of entries are possible in real-life only if a total lack of knowledge concerning the occurring attribute value (except its existence) takes place – which may raise valid questions about the point of analyzing such data. It seems practical to consider interpreting such highly imprecise entries, in the same way a data analyst would act when encountering a *null* value in the relational database table. Especially in the context of approaching a *null* value reflecting *unknown interpretation* [26–29] (as opposite to the *null* being interpreted as *nonexistent* value).

The theoretical investigation on the worst-case scenario brings us also to the computational efficiency of using dendrograms (see *G8* in Fig. 4 for an example) to represent similarities between different attribute values. These hierarchies, by their tall nature, may lead to the most costly searches when we use DFS-based algorithms. Dendrogram is the tree, whose height is equal to the number of attribute values (i.e. $H = \|A^0\|$ – no higher partition tree can exist). It represents the most extreme case of the partition tree, where an intra-level granularity (e.g. $\|A^h\|$) can differ from the granularity of the neigh-

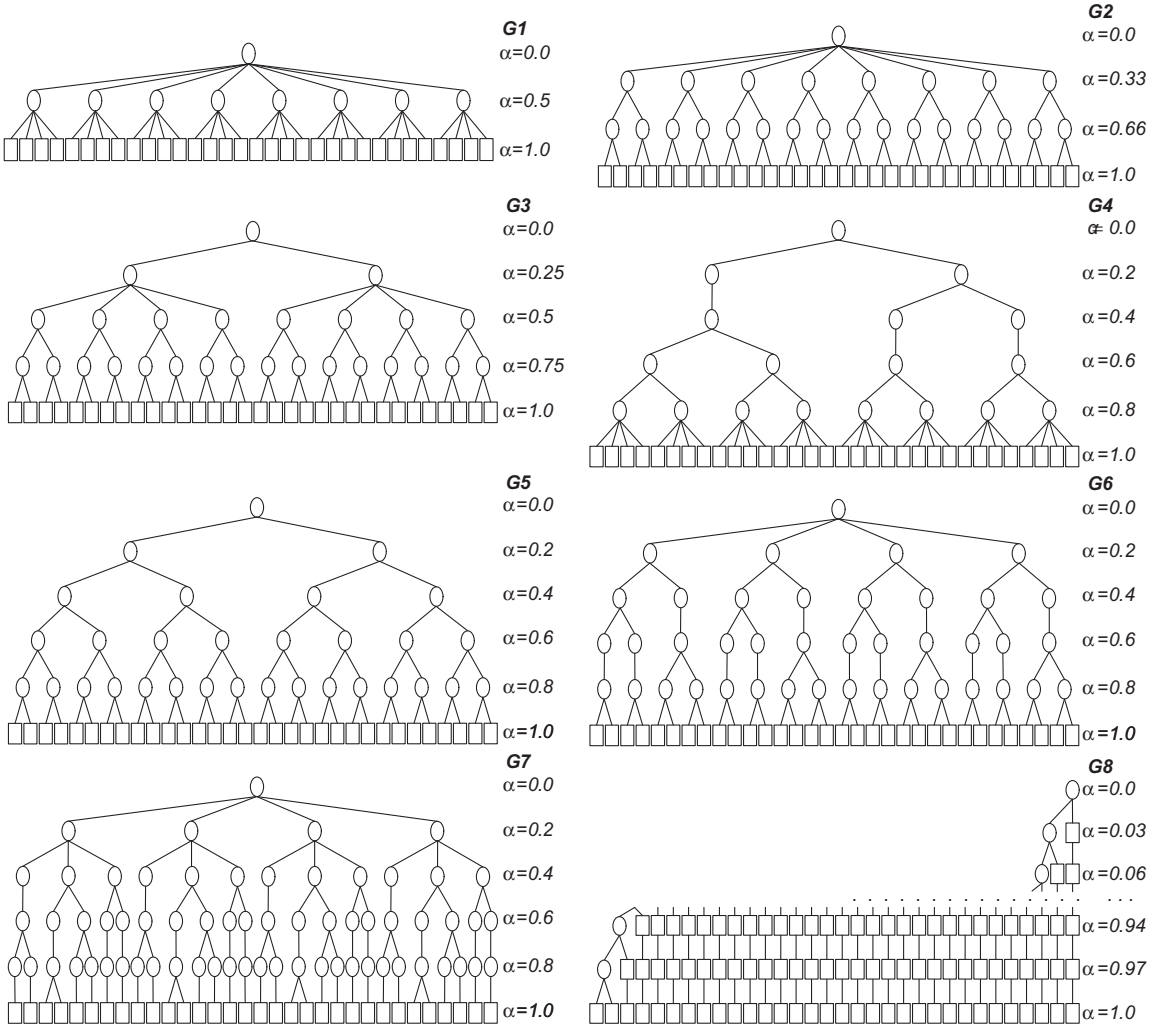

Fig. 4. Different shapes of partition trees used during our tests.

boring levels, only by one (i.e. $\|A^{h-1}\| - 1 = \|A^h\| = \|A^{h+1}\| + 1$). The dendrogram also maintains the largest possible difference between b and H values (i.e. $b \ll H$), and contains the largest possible number of vertices (i.e. equivalence classes): $V = \sum_{h=0}^{H-1} (\|A^0\| - h) = (\|A^0\| + 1) \cdot \frac{H}{2}$. We will show some experimental results showing the costs of searching through dendrograms in the next section.

Obviously, the second scenario (i.e. $O(H)$) occurs when the candidate for subset of resemblances (i.e. our c_i in Table 3) is repeatedly found only in one descendent of every node along the traversal trail starting from the root and finishing at the $\alpha = 1.0$ level. This is related to the concept of "the probability parameter that characterizes the search" [37], mentioned at the beginning of this section. Let us think about it a little. If we have a record that contains only one entry (no imprecision!) and still decide to apply our algorithm on it, our *DFSearchForResemblances* would cut off all irrelevant subtrees at every level of our partition tree, generating only single search branch from the top of the tree to its bottom. This branch has the height of H, and this is where $O(H)$ comes from – after all, this is what makes the usage of trees so effective in large-scale systems.

It may be surprising to learn that similar situation may occur when searching the tree using multiple values. If imprecise tuples contain multiple attribute values, that were put by domain experts in the same equivalence class through multiple levels of the partition tree, the *DFSearchForResemblances* is going to be completed with the same $O(H)$ time complexity. Of course, the higher the data imprecision (i.e. the number of values entered to the record), the smaller is the probability of such phenomena occurring (unless of course, the large equivalence class, matching these frequently entered values, was placed on multiple levels of the partition tree). This part suggests that our experimental evaluation should not only investigate the scalability of our approach in the context of the size of the fuzzy data table (i.e. the number of records), but also from the perspective of data imprecision.

On more general level, we can draw a conclusion that the performance of our *DFSearchForResemblances* procedure is dependent on how well the expert knowledge about the data domain (reflected by the pre-defined fuzzy similarity relation) matches the actual behavior of the data (reflected by character of imprecision, i.e. by the number of non-atomic values and their distribution within the given data entry).

To conclude our theoretical investigation, we want to point out that our lossy specialization algorithm is repeated for each of the imprecise database records, what brings us back to the first of the three factors mentioned at the beginning of this section. The repetitive application of our algorithm increases its cost by the number of imprecise database records, which we will denote by N. This will further increase the total time complexity of our approach by the $O(N)$ factor. In large-scale database systems it is safe to assume that number of records, (N) is much bigger than the height of the partition tree (H), or even the number of its vertices ($V = b^H$). For instance, the domain with $\|A^0\| = 32$ attribute values may have a partition tree not higher than $H = 32$ levels (i.e. dendrogram), that contains no more than a total of $V = (32 + 1) \cdot \frac{32}{2} = 528$ nodes; whereas it can easily generate $2^{32} = 4{,}294{,}967{,}296$ unique imprecise entries. If we assume the following proportions: $H \ll b^H \ll N$ (e.g. 32 levels of dendrogram $\ll$ 528 vertices in the dendrogram $\ll$ 50,000 imprecise records, as presented in the next section with experiments), it is safe to approximate the time complexity of our lossy specialization algorithm with $O(N)$.

As the reader probably observed by now, this theoretical investigation has a multi-dimensional character, where the dimensions of our analysis are derived from the three factors we listed at the beginning of this section. This discussion shows well the importance of an experimental average case analysis, which we will present in the next section.

3.5. Time complexity – experimental results

To objectively evaluate the performance of our similarity-driven specialization algorithm, we conducted some experiments using artificially generated data sets, with a large number of highly-imprecise fuzzy database records, and eight different types of partition trees (see Fig. 4). The test data were generated by randomly picking values from the domain containing 32 symbolic values of equal size, interpreted as distinct equivalence classes at the lowest (i.e. $\alpha = 1.0$) level of our all partition trees. We picked 32 values, to be able to easily present shapes and parameters of our partition trees to the reader. The partition trees in Fig. 4 were ordered based on the number of levels, and their detailed parameters: (1) height (denoted as H), (2) number of vertices (V), and (3) average branching factor (calculated as $b = \sqrt[H]{V}$), are presented in Table 4.

G1 is the smallest partition tree, it has $H = 3$ levels, and 41 nodes. *G4–G7* have all equal height ($H = 6$), but their branching factors (b's) differ. It is important to note that the branching for individual levels in the *G4–G7* hierarchies is distributed differently. For instance, *G5* has a branching factor constant in all of its levels. The number of branches in *G4* is low at the top and increases as we move down the hierarchy. *G7* shows an opposite tendency (high branching occurs close to the root, and decreases as we traverse toward the bottom), which increases the total number of nodes in this tree ($V_{G7} > V_{G4}$). The last par-

Table 4
Parameters of Hierarchies from Fig. 4.

Partition Tree	Number of vertices (V)	Number of levels (H)	Avg. branching (b)	Partition Tree	Number of vertices (V)	Number of levels (H)	Avg. branching (b)
G1:	*41*	*3*	*3.45*	***G2:***	*57*	*4*	*2.75*
G3:	*59*	*5*	*2.26*	***G4:***	*50*	*6*	*1.92*
G5:	*63*	*6*	*1.99*	***G6:***	*73*	*6*	*2.04*
G7:	*97*	*6*	*2.14*	***G8:***	*528*	*32*	*1.22*

tition tree, *G8*, reflects a dendrogram (i.e. the tallest partition tree that we were able to create for domain of $\|A^0\|$ = 32 values). Before discussing our experimental results, we want to point reader's attention to the fact that when ordered by the total number of vertices, the hierarchies would be placed in the following order: *G1*, *G4*, *G2*, *G3*, *G5*, *G6*, *G7*, and *G8*. Due to the small number of branches at the top of the tree, *G4* has fewer nodes than shorter trees (i.e. *G2*, *G3*) with higher branching factors. This information may come useful when analyzing our experimental results, shown in Figs. 5–7.

Non-atomic values in a fuzzy tuple (stored during our specialization process in the Imprecise Attribute Entry, denoted as *T* in Table 3) have been considered as 75% imprecise (see Table 5), when the number of unique values in the tuple was at 75% of the domain size at the lowest abstraction level (i.e. 24 values for a domain containing $\|A^0\|$ = 32 values). To simulate random distribution of imprecision, we have chosen these 24 values in random order for each tuple of the 75%-imprecise data set (note that the distribution of non-atomic values in fuzzy tuples, used during our tests, does not follow the distribution of equivalence classes in any of the used partition trees). Although our simplified measure of data imprecision is clearly imperfect (as existence of only atomic values in the data, would still be interpreted as 1/32 = 3.125% of imprecision, which is obviously in conflict with the interpretation of atomic values as entirely precise), we decided to accept this inaccuracy to simplify the presentation of our experimental results.

The number of records in each data file used for imprecision-dependent evaluation was 50,000. The large number of imprecise records, with random distribution of values, was generated to ensure that our results reflect an average case traversal, which we wanted to have when comparing our experiments. The algorithms' average running times for different degrees of imprecision in data sets, for different shapes of partition trees (presented in Fig. 4), are shown in Fig. 5. To conduct comparable experiments we wanted to preserve the original randomness in our imprecise data sets when running all of our experiments. Therefore, to generate the 68.75%-imprecise data file we simply eliminated two randomly chosen values from each of our 75%-imprecise records. The random removal of two values was continued until we had only records containing atomic values (marked as 3.125% in Fig. 5).

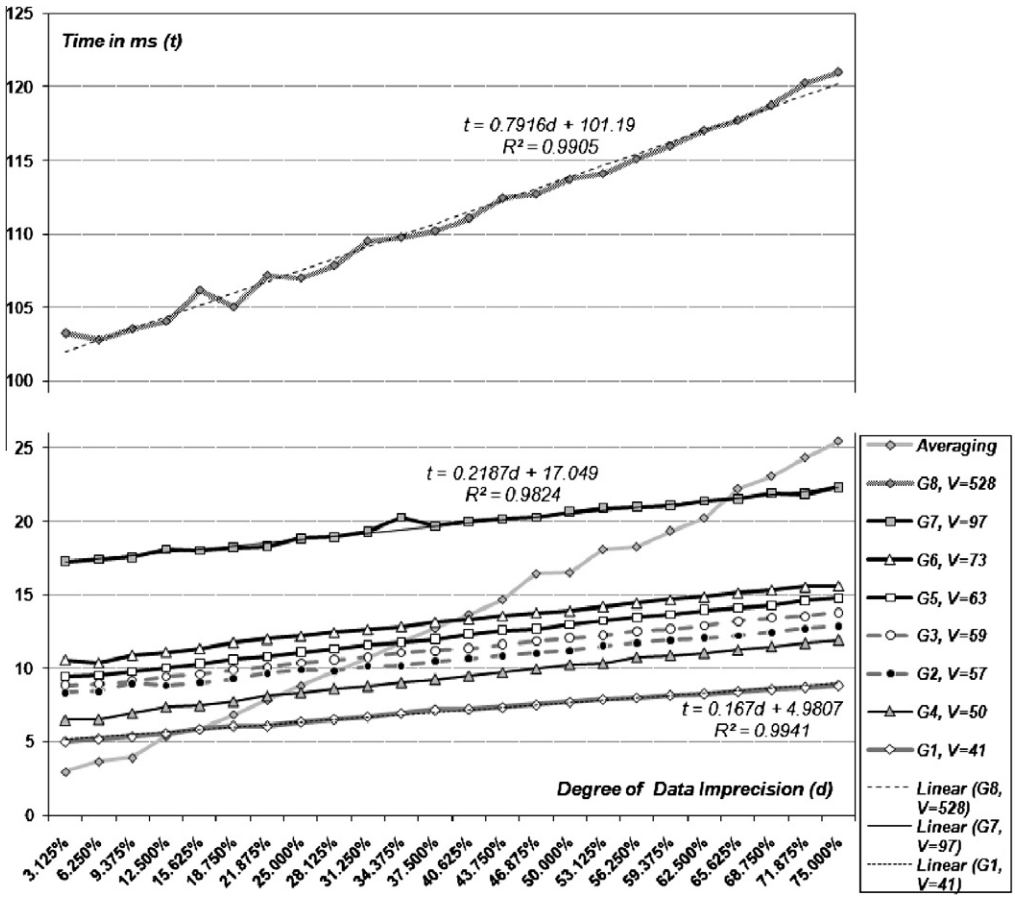

Fig. 5. Total running times to interpret 50,000 of fuzzy records as a function of data imprecision. Results were generated using different types of partition trees (see Fig. 4), and fuzzy data tables with different imprecision levels (see Table 4 for examples). The charts for *G1*, *G7*, and *G8* contain linear trend lines, with the values of functions' equations and *R*-squared errors (R^2) calculated.

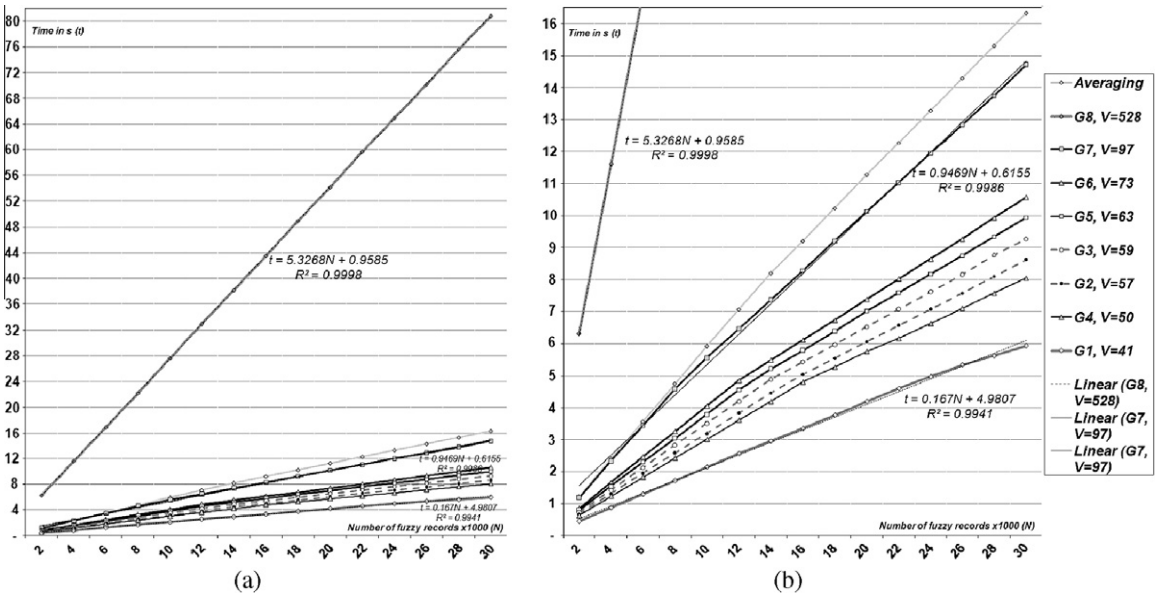

Fig. 6. Algorithm's scalability for 75% of data imprecision. (a) Total running times, taken by our algorithm for all partition trees presented in Fig. 4. (b) Close-up for partition trees *G1*–*G7*. The charts for *G1*, *G7*, and *G8* contain linear trend lines, with the values of functions' equations and *R*-squared errors (R^2) calculated.

Before moving further with our investigation, we want our reader to have a careful look at the results presented in Fig. 5. As expected for DFS-based implementation, our algorithm performed its worst on the tallest hierarchy (*G8*), and its best on the hierarchy with the smallest number of levels (*G1*). Depth-first traversing of dendrogram-like partition trees (where $b \ll H$) becomes especially time consuming for highly imprecise data sets. This suggests that when given a task of large, highly

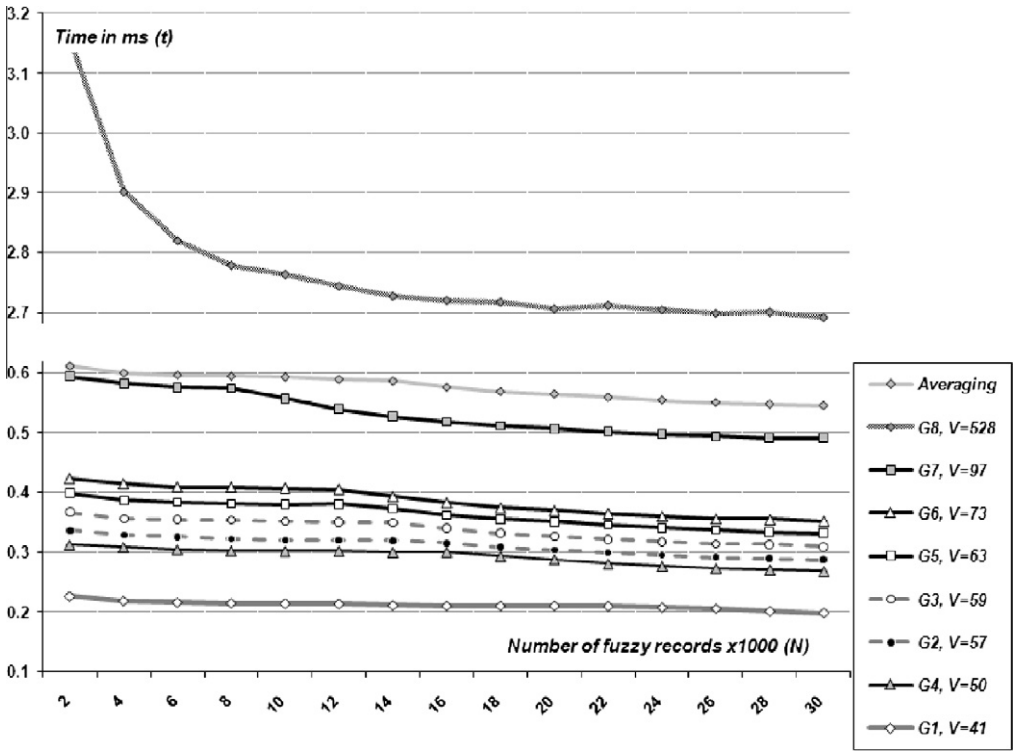

Fig. 7. Algorithm's scalability for 75% of data imprecision – average times, reflecting time needed to interpret a single fuzzy record.

Table 5
Imprecise tuples from data files with different imprecision levels.

Degree of data imprecision (%)	Example of *imprecise attribute entry* (*T*) for domain of 32 symbolic values
75	*a, d, g, j, m, p, s, w, z, b, e, h, k, n, q, t, x, c, f, i, l, o, r, u*
68.75	*d, g, j, m, p, s, w, z, b, e, h, k, n, q, t, x, c, f, i, l, o, r*
62.5	*d, g, m, p, s, w, z, b, e, h, k, n, q, t, x, c, f, i, l, r*
56.25	*d, g, m, p, s, w, b, e, h, k, n, q, t, x, c, f, i, l*
50	*g, m, p, s, w, b, e, h, k, n, q, t, c, f, i, l*
43.75	*g, m, p, s, w, b, h, k, n, q, t, c, f, l*
37.5	*g, m, p, s, b, h, k, n, q, c, f, l*
31.25	*m, p, s, b, k, n, q, c, f, l*
25	*m, p, s, b, n, q, c, f*
18.75	*p, s, n, q, c, f*
12.5	*p, s, q, c*
6.25	*s, q*
3.125	*g*

imprecise data set interpretation, we may want to consider flattening the partition trees before applying our heuristic algorithm for the fuzzy tuples' interpretation. For some good pointers on how to flatten dendrograms, derived from data using hierarchical agglomerative clustering (HAC) algorithms, to more bushy structures with smaller number of levels (which resemble more structures defined by human experts), see [30–32].

The results generated using partition tree *G4* are also very interesting. This tree generated the second-best performance, separating itself by a large margin from the other trees with the same number of levels (compare the timing for *G4* with the plots generated using *G5–G7*). This confirms our theoretical investigation, which concluded that, when having everything else constant (e.g. the number of records and data imprecision), the partition trees with the smaller number of nodes lead to better results. Trees that have the same numbers of levels (e.g. *G4–G7*) are going to contain a bigger number of vertices if they have a higher branching factor at the top and therefore will take longer (on average) to be searched through. As it can be observed from the trend lines, presented in the slope-intercept forms in Fig. 5, both the time-intercept (needed to initialize data structures in the memory), and the slope (reflecting mainly the costs of searching through the trees) grow with the size of partition trees. The *R*-squared error is the square of the correlation coefficient, that shows correlation between the experimental results and the trend line. A value of $R = 1$ indicates an exact linear relationship between the degree of imprecision and running time. Our values of *R*, which are very close to 1, indicate a good linear reliability.

Following a request from one of our reviewers, we compared our results against a simple *Averaging* strategy (mentioned in Section 3.2), which counts the number of values entered into the tuple and splits its vote between all of the inserted values evenly. As it can be observed from Fig. 5, both algorithms have a linear time growth, when compared in the context of data imprecision.

In the second part of experimental evaluation, we include a scalability assessment of our algorithm. For this task, we have chosen the most imprecise of our data sets (i.e. 75%, where all records contain 24 descriptors) and kept gradually reducing size of our fuzzy data table (we started with 30,000 records and finished with 2000 records). As it can be seen from the results presented in Fig. 6, the algorithm has linear characteristics, which is very important for estimating the time needed to interpret large data sets with a constant (or at least bounded from the top) degree of imprecision. Once again, we present a simple *Averaging* as a benchmark for our approach – our experiments show that both algorithms have $O(N)$ growth, when $N \gg V \gg H$ – which is a typical case for database systems.

Fig. 7 contains average times for individual records. As expected, the most expensive is the search through the dendrogram (*G8*). Significantly high average time for the smallest data set (2000 records) shows well the constant cost related to loading the dendrogram-like hierarchies to the primary memory. The significance of this overhead gets reduced when the average times for larger data sets are generated.

4. An example of using lossy specialization for decision tree-based classification

After the evaluation of computational costs of our approach, we would like to show the applicability of fuzzy tuples' specialization to the data mining process. In this section we are going to show how fuzzy records can be incorporated into a classic entropy-based classification process (e.g. ID3 [5], C4.5 [17]). We will present an illustrative example of generating decision tree from imprecise tuples, which involves multi-attribute data specialization.

Our fuzzy database table is presented in Table 6. Our example is a modified version of a non-fuzzy problem used by Russell and Norvig to explain induction of decision trees, in their well-known textbook on artificial intelligence [33] (the example we are referring to can be found on pp. 653–658).

Let us assume we gathered data from different clients regarding their decision about waiting (or not) for a table at a restaurant. Some of the clients did not exactly remember the situations they were reporting on, so we used non-atomic values to reflect such lack of certainty. Our data table has four attributes, i.e.: *Waiting Time, Day of Week, Food Type, Did Wait?*. To

Table 6
Fuzzy database relation on consumers' behaviors in restaurants (25 tuples).

ID	Waiting time	Day of week	Food type	Did wait?
1	*30–60*	*Thr*	*Greek*	*No*
2	*10–30*	*Fri*	*Italian*	*Yes*
3	*30–60, Above 60*	*Thr*	*Burger, Barbeque*	*No*
4	*30–60*	*Fri, Sat*	*Greek*	*Yes*
5	*Above 60*	*Tue*	*Barbeque, Burger*	*No*
6	*10–30*	*Tue*	*Burger, Barbeque*	*Yes*
7	*30–60*	*Mon*	*Italian*	*No*
8	*30–60*	*Fri, Sun*	*Burger*	*Yes*
9	*0–10*	*Fri*	*Italian, Barbeque, Greek*	*Yes*
10	*30–60*	*Sun*	*Chinese*	*No*
11	*30–60*	*Mon, Tue*	*Barbeque*	*No*
12	*10–30*	*Mon*	*Greek*	*No*
13	*30–60*	*Fri*	*French, Italian*	*Yes*
14	*30–60*	*Fri, Sat*	*Barbeque*	*Yes*
15	*10–30*	*Sun*	*Barbeque, Burger*	*Yes*
16	*30–60*	*Sun*	*French, Greek, Italian*	*No*
17	*30–60*	*Wed, Thr*	*French*	*No*
18	*10–30*	*Thr*	*French*	*Yes*
19	*0–10*	*Fri, Sat, Sun*	*Barbeque*	*Yes*
20	*30–60, Above 60*	*Sun, Mon, Tue*	*Sushi*	*No*
21	*10–30*	*Wed*	*Chinese*	*No*
22	*30–60*	*Fri*	*Chinese, Sushi*	*No*
23	*10–30*	*Sat*	*Sushi, Chinese*	*Yes*
24	*30–60*	*Sun*	*Barbeque*	*Yes*
25	*0–10*	*Wed*	*Chinese, Sushi*	*Yes*

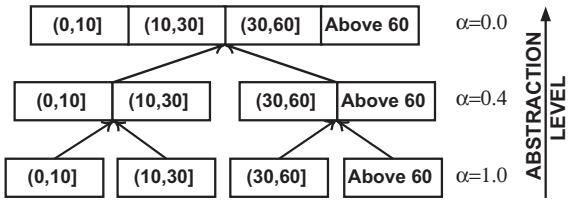

Fig. 8. Partition tree for the domain *Waiting Time*.

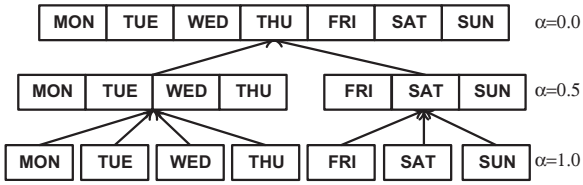

Fig. 9. Partition tree for the domain *Day of Week*.

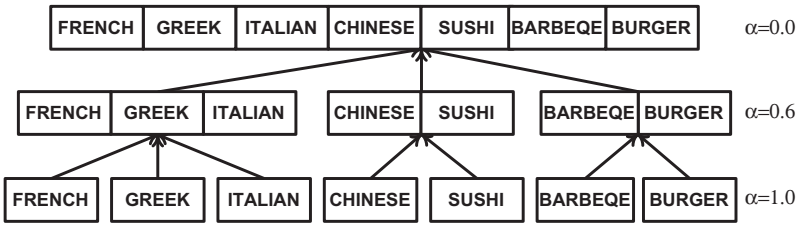

Fig. 10. Partition tree for the domain *Food Type*.

keep our example short the last (i.e. class label) attribute has a non-fuzzy character that answers the question whether a particular client decided to wait or not for a table (i.e. there are only two labels in our classification example: *Yes* and

Table 7
Part of fuzzy relation after complete data specialization.

ID	Wait time	Day of week	Food type	Did wait?	**COUNT**
9a	*0–10*	*Fri*	*Italian*	*Yes*	*0.38*
9b	*0–10*	*Fri*	*Barbeque*	*Yes*	*0.24*
9c	*0–10*	*Fri*	*Greek*	*Yes*	*0.38*
⋮	⋮	⋮	⋮	⋮	⋮
19a	*0–10*	*Fri*	*Barbeque*	*Yes*	*0.333*
19b	*0–10*	*Sat*	*Barbeque*	*Yes*	*0.333*
19c	*0–10*	*Sun*	*Barbeque*	*Yes*	*0.333*

No), when the circumstances described in the remaining three attributes were taking place. By looking at the Table 6 we can notice that 17 out of 25 records contain some degree of imprecision.

As expected when using a fuzzy relational database, the similarity relations have been specified for our attributes. To make our example more understandable, we present the fuzzy similarity relations using partition trees. Fig. 8 presents the partition tree for the domain *Waiting Time*. Fig. 9 presents the partition tree for the attribute *Day of Week*, and Fig. 10 reflects the fuzzy similarity relation for the *Food Type*.

As presented in the Section 3.3, we are capable of transferring the original fuzzy database table (Table 6), to a data table that contains only atomic variables, where the attribute COUNT is added to reflect original proportions between the data items. We used our *Similarity-Driven Complete Data Specialization* algorithm (Table 3), to transform our data table to a 1NF data table which contains distribution of records' fractions based on the pre-defined similarities between the originally entered attribute values.

Let us look at the imprecise tuple number 9 in Table 6 (our fuzzy database table), i.e. {*0–10; Fri; Italian, Barbeque, Greek; Yes*}, the distribution of the COUNT for this tuple, after its *complete defuzzification* (to atomic descriptors at the $\alpha = 1.0$ level)

```
Wait_Time = 0-10: Yes (3.02)
Wait_Time = 10-30
|   Day_Of_Week = Mon: No (1.00)
|   Day_Of_Week = Tue: Yes (1.00)
|   Day_Of_Week = Wed: No (1.00)
|   Day_Of_Week = Thr: Yes (1.00)
|   Day_Of_Week = Fri: Yes (1.00)
|   Day_Of_Week = Sat: Yes (1.00)
|   Day_Of_Week = Sun: Yes (1.00)
Wait_Time = 30-60
|   Day_Of_Week = Mon: No (1.70)
|   Day_Of_Week = Tue: No (0.70)
|   Day_Of_Week = Wed: No (0.50)
|   Day_Of_Week = Thr: No (2.02)
|   Day_Of_Week = Fri
|   |   Food_Type = Barbeque: Yes (0.50)
|   |   Food_Type = Burger: Yes (0.50)
|   |   Food_Type = French: Yes (0.50)
|   |   Food_Type = Greek: Yes (0.50)
|   |   Food_Type = Italian: Yes (0.50)
|   |   Food_Type = Chinese: No (0.50)
|   |   Food_Type = Sushi: No (0.50)
|   Day_Of_Week = Sat: Yes (1.00)
|   Day_Of_Week = Sun
|   |   Food_Type = Barbeque: Yes (1.00)
|   |   Food_Type = Burger: Yes (0.50)
|   |   Food_Type = French: No (0.34)
|   |   Food_Type = Greek: No (0.33)
|   |   Food_Type = Italian: No (0.33)
|   |   Food_Type = Chinese: No (1.00)
|   |   Food_Type = Sushi: No (0.12)
Wait_Time = Above-60: No (2.04)
```

Fig. 11. Decision tree generated from completely specialized data set using ID3-type algorithm.

using our data specialization algorithm, is presented in the three upper rows of Table 7. There is more emphasis on *Italian* and *Greek* entries, since these two values are more similar according to our fuzzy similarity relation reflected in Fig. 10.

Results of the complete data specialization, preformed on the tuple with *ID* = 19 in Table 6, are distributed over the three next rows shown in the Table 7. This time the tuple's vote is distributed evenly, since *Fri*, *Sat*, and *Sun* are considered to be equally similar on all levels of abstraction in Fig. 9. As we can see from Table 7, complete defuzzification of the original data set may let us merge certain parts of records, which represent the fractions of originally different fuzzy tuples that contain identical atomic values after the defuzzification process is completed. In the examples presented in Table 6, the parts of tuples identified as *9b* and *19a* contain identical descriptors, and can be merged for classification purposes to the form: {0–10; *Fri; Barbeque; Yes*; 0.573}. Note that the value in the COUNT attribute has been appropriately updated (i.e. 0.24 + 0.333 = 0.573) to maintain the original proportions of values' distribution within the data set.

It is worth performing merging of identical defuzzified tuples before running data mining algorithms, as reduced length of the data table allows a speed up processing time of these (usually expensive) algorithms. It is important to remember that COUNT of the merged tuples needs to be updated appropriately (this is to make sure that the original proportions between data entries are maintained through the whole data mining process).

Fig. 11 includes a decision tree generated by the ID3 algorithm on completely (i.e. to $\alpha = 1.0$ level, for all of the attributes) specialized data. The tree has a total of 32 nodes, where 28 leaves accurately classify all the data. Numbers in bracket, printed after the leaves, represent fractions of vote that were assigned to the particular leaf/class.

5. Conclusions

The primary purpose of this work was to develop an algorithm that allows for a background knowledge-based transition of the imprecise data, which we can store in similarity-based fuzzy relational databases, to a format that can be easily analyzed by broadly available software for data analysis (e.g. Weka [34], Matlab [35]). By proposing this heuristics we hope to open a discussion among researchers working in fuzzy database community on the time-efficient algorithms allowing intuitive transition of large, imprecise data collections to the formats compatible with the first normal form [4]. We strongly believe that, by the incorporation of domain-specific background knowledge directly to the imprecise data interpretation, many useful algorithms can be developed, putting fuzzy databases on the map for the quickly growing data mining community.

In this paper, we intentionally focused on general considerations and used artificially generated data sets, trying to avoid making this presentation application specific. This philosophy, however, carries some important limitations – all of our conclusions need to be interpreted carefully as they were derived using data samples with randomly distributed imprecision. In real life, we would expect distribution of non-atomic values in the fuzzy data records to follow closely the layout of equivalence classes in the fuzzy similarity relation, which characterizes the related domain. This may, however, generate even better results by improving efficiency of our algorithm's DFS component.

Acknowledgments

Rafal Angryk would like to thank the Montana NASA EPSCoR Grant Consortium for sponsoring a part of this research (Award No. M166-05-Z3184). Some parts of the experimental results were generated using the code developed by Mr. Shahriar Hossain.

We would like to also express our gratitude to anonymous reviewers, whose questions helped to improve our presentation. We are deeply thankful for their work.

References

[1] R. Angryk, F. Petry, R. Ladner, Mining generalized knowledge from imperfect data, in: Proceedings of the 10th Internatinal Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU '04), Perugia, Italy, July 2004, pp. 739–746.
[2] R. Angryk, Similarity-driven defuzzification of fuzzy tuples for entropy-based data classification purposes, in: Proceedings of the 15th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE '06), Vancouver, Canada, July 2006, pp. 1490–1498.
[3] J. Gray, Talks at the HP Labs/MSR (July 2004), and at the University of Tokyo (October 2005), URL: <research.microsoft.com/~gray/talks/Info%20Avalanche%20U%20Tokyo%20.ppt>.
[4] E.F. Codd, A relational model of data for large shared data banks, Commun. ACM 13 (6) (1970) 377–387.
[5] X. Wu, V. Kumar, R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G.J. McLachlan, A. Ng, B. Liu, P.S. Yu, Z. Zhou, M. Steinbach, D.J. Hand, D. Steinberg, Top 10 algorithms in data mining, Knowl. Inform. Syst. 14 (1) (2007) 1–37.
[6] B.P. Buckles, F.E. Petry, A fuzzy representation of data for relational databases, Fuzzy Sets Syst. 7 (3) (1982) 213–226.
[7] B.P. Buckles, F.E. Petry, Information-theoretic characterization of fuzzy relational databases, IEEE Trans. Syst. Man Cybern. 13 (1) (1983) 74–77.
[8] F.E. Petry, Fuzzy Databases: Principles and Applications, Kluwer Academic Publishers, Boston, MA, 1996.
[9] L.A. Zadeh, Similarity relations and fuzzy orderings, Inform. Sci. 3 (2) (1970) 177–200.
[10] S. Kumar De, R. Biswas, A.R. Roy, On extended fuzzy relational database model with proximity relations, Fuzzy Sets Syst. 117 (2001) 195–201.
[11] S. Parsons, Current approaches to handling imperfect information in data and knowledge bases, IEEE Trans. Knowl. Data Eng. 8 (3) (1996) 353–372.
[12] Z. Ma, Fuzzy database modeling with XML, in: Series: Advances in Database Systems, vol. 29, XXIV, Springer-Verlag, Berlin, 2005.
[13] Z. Ma, Fuzzy database modeling of imprecise and uncertain engineering information, in: Series: Studies in Fuzziness and Soft Computing, vol. 195, XV, Springer-Verlag, Berlin, 2006.
[14] P. Bertrand, F. Goupil, Descriptive statistics for symbolic data, in: H.-H. Bock, E. Diday (Eds.), Analysis of Symbolic Data: Exploratory Methods for Extracting Statistical Information from Complex Data, Springer-Verlag, Berlin, 2000, pp. 103–124.

[15] L. Billard, E. Diday, Symbolic Data Analysis: Conceptual Statistics and Data Mining, Wiley Series in Computational Statistics, Wiley, 2007.
[16] R. Angryk, F. Petry, Mining multi-level associations with fuzzy hierarchies, in: Proceedings of the 14th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE '05), Reno, NV, USA, May 2005, pp. 785–790.
[17] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kauffman, San Francisco, CA, 1993.
[18] S.L. Crawford, Extension to the CART algorithm, Int. J. Man–Machine Stud. 31 (8) (1989) 197–217.
[19] U.M. Fayyad, K.B. Irani, The attribute selection problem in decision tree generation, in: Proceedings of National Conference on Artificial Intelligence (AAAI'90), pp. 104–110.
[20] J. Han, Y. Cai, N. Cercone, Knowledge discovery in databases: an attribute-oriented approach, in: Proceedings of 18th International Conference on Very Large Data Bases (VLDB '92), Vancouver, Canada, 1992, pp. 547–559.
[21] J. Han, Y. Cai, N. Cercone, Data-driven discovery of quantitative rules in relational databases, IEEE Trans. Knowl. Data Eng. 5 (1) (1993) 29–40.
[22] D.H. Lee, M.H. Kim, Database summarization using fuzzy ISA hierarchies, IEEE Trans. Syst. Man Cybern. Part B 27 (1) (1997) 68–78.
[23] G. Raschia, L. Ughetto, N. Mouaddib, Data summarization using extended concept hierarchies, in: Proceedings of Joint Ninth IFSA World Congress and 20th NAFIPS International Conference, Vancouver, BC, Canada, 2001, pp. 2289–2294.
[24] R. Angryk, F. Petry, Consistent fuzzy concept hierarchies for attribute generalization, in: Proceedings of Second International Conference on Information and Knowledge Sharing (IKS '03), Scottsdale, AZ, USA, November 2003, pp. 158–163.
[25] R. Angryk, F. Petry, Knowledge discovery in fuzzy databases using attribute-oriented induction, in: T.Y. Lin, S. Ohsuga, C.J. Liau, X. Hu (Eds.), Foundations and Novel Approaches in Data Mining, in: Series: Studies in Computational Intelligence, vol. 9, X, Springer-Verlag, Berlin, 2006, pp. 169–196.
[26] D. Popat, H. Sharda, D. Taniar, Classification of fuzzy data in database management system, in: Proceedings of Eighth International Conference on Knowledge-based Intelligent Information and Engineering Systems (KES '04), Wellington, New Zealand, September 2004, in: Series: Lecture Notes in Artificial Intelligence (LNAI), vol. 3214, Springer-Verlag, 2004, pp. 691–697.
[27] K. Selcuk Candan, J. Grant, V.S. Subrahmanian, A unified treatment of null values using constraints, Inform. Sci. 98 (1–4) (1997) 99–156.
[28] C. Zaniolo, Database relations with null values, in: Proceedings of ACM Symposium on Principles of Database Systems (PODS '82), March, Los Angeles, CA, 1982, pp. 27–33.
[29] S. McClean, B. Scotney, M. Shapcott, Using background knowledge with attribute-oriented data mining, in: Proceedings of IEE Colloquium on Knowledge Discovery and Data Mining, London, UK, May 1998, pp. 1–4.
[30] S.-L. Chuang, L.-F. Chien, Towards automatic generation of query taxonomy: a hierarchical query clustering approach, in: Proceedings of Second IEEE International Conference on Data Mining (ICDM-IEEE '02), Maebashi City, Japan, December 2002, pp. 75–82.
[31] S.-L. Chuang, L.-F. Chien, A practical Web-based approach to generating topic hierarchy for text segments, in: Proceedings of Conference Information and Knowledge Management (CIKM'04), Washington, DC, November 2004, pp. 127–136.
[32] B. Wall, N. Richter, R. Angryk, Creating concept hierarchies in an information retrieval system, in: Proceedings of Fifth IEEE International Conference Data Mining (ICDM-IEEE '05), Workshop on Foundations of Semantic Oriented Data and Web Mining, Houston, TX, USA, November 2005, pp. 99–105.
[33] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, second ed., Prentice Hall, 2002 (the example we are referring to in the paper can be found on pp. 653–658).
[34] Weka URL: <http://www.cs.waikato.ac.nz/ml/weka/>.
[35] Matlab URL: <http://www.mathworks.com/products/matlab/>.
[36] R.M. Karp, J. Pearl, Searching for an optimal path in a tree with random costs, Artif. Intell. 21 (1983) 99–117.
[37] H.S. Stone, P. Sipala, The average complexity of depth-first search with backtracking and cutoff, IBM J. Res. Develop. 30 (3) (1986) 242–258.