Chapter 12
Fuzzy Cardinalities as a Basis to Cooperative Answering

Grégory Smits, Olivier Pivert and Allel Hadjali

Abstract Cooperative approaches to relational database querying help users retrieve the tuples that are the most relevant with respect to their information needs. In this chapter we propose a unified framework that relies on a fuzzy cardinality-based summary of the database. We show how this summary can be efficiently used to explain failing queries or to revise queries returning a plethoric answer set.

1 Introduction

The paradigm of cooperative answering is originated from the works in the context of natural-language question-answering done by Kaplan [20] in the end of the seventies. One of the aims of such works is to avoid natural-language query systems to produce "there is zero result" when a query fails. Cooperative intelligent systems should rather correct any false presupposition of the user, anticipate follow-up queries and provide information not explicitly requested by the user.

Cooperative responses to a query are indirect responses that are more helpful to the user than direct, literal responses would be. Interest in cooperative responses in the database field arises in the middle of the eighties [13, 14, 17, 25, 31]. In this context, cooperative answering represents intensional, qualified or approximate answers. They may explain the failure of a query to produce results, build queries that are related to the original one and re-submit them for an evaluation. Most cooperative

G. Smits (✉)
IRISA-IUT, Lannion, France
e-mail: gregory.smits@univ-rennes1.fr

O. Pivert · A. Hadjali
IRISA-ENSSAT, Lannion, France
e-mail: pivert@enssat.fr

A. Hadjali
e-mail: hadjali@enssat.fr

O. Pivert and S. Zadrożny (eds.), *Flexible Approaches in Data, Information and Knowledge Management*, Studies in Computational Intelligence 497,
DOI: 10.1007/978-3-319-00954-4_12,

techniques proposed in the literature deal with the empty answer problem in a crisp query setting.

In this chapter, we consider fuzzy queries which express preferences modeled using fuzzy set membership functions (that describe the preference profiles of the user on each attribute domain involved in the query). We address two problematic situations users can be faced with when querying relational databases: their query returns (i) an empty set of answers or, (ii) a plethoric answer set. We propose a uniform solution to these two symmetrical problematic situations. This solution relies on the precomputation of a summary of the queried database according to a predefined shared vocabulary. This summary provides information about the distribution of the data over the definition domains of the different target attributes. This summarization strategy efficiently computes fuzzy cardinalities using a single scan of the database.

Recall that with respect to Boolean queries, fuzzy queries reduce the risk of obtaining an empty set of answers since the use of a finer discrimination scale—[0, 1] instead of {0, 1}—increases the chance for an element to be considered somewhat satisfactory. Nevertheless, the situation may occur where none of the elements of the target database satisfies the query even to a low degree.

In the context of fuzzy queries, beside the empty answer set (EAS) problem, another situation deserves attention: that where the answer set is not empty but only contains elements which satisfy to a *low degree* the preferences specified in the user query. We show in this chapter that a generic—and very efficient—type of approach that leverages fuzzy cardinalities may be employed to provide explanations for both types of situations (empty or unsatisfactory answer set). Minimal failing subqueries [24] constitute useful explanations about the conflicts in a failing query. These explanations may (i) help the user revise or reformulate his/her initial query or (ii) be used to set up an automatic and targeted relaxation strategy.

As for the symmetrical problem, i.e. the plethoric answer set (PAS) problem, it has been intensively addressed by the information retrieval community and two main approaches have been proposed for Boolean queries. The first one, that may be called data-oriented, aims at ranking the answers in order to return the best k ones to the user. However, this strategy is often faced with the difficulty of comparing and distinguishing among tuples that satisfy the initial query. In this data-oriented type of approach, we can also mention works which aim at summarizing the answer set to a query [36].

The second type of approach may be called query-oriented as it performs a modification of the initial query in order to make it more selective. For instance, a strategy consists in strengthening the specified predicates (as an example, a predicate $A \in [a_1, a_2]$ becomes $A \in [a_1 + \gamma, a_2 - \gamma]$) [6]. However, for some predicates, this strengthening (if applied in an interated way) can lead to a deep modification of the meaning of the initial predicate. Another type of approach advocates the use of user-defined preferences on attributes which are not involved in the initial query [3, 12, 21]. Such a subjective knowledge can then be used to select the most preferred items among the initial answer set. Still another category of query-oriented approaches [26, 27] aims at automatically completing the initial query with

additional predicates to make it more demanding. Our work belongs to this last family of approaches but its specificity concerns the way additional predicates are selected.

Indeed, we consider that the predicates added to the query must respect two properties: (i) they must reduce the size of the initial answer set, (ii) they must modify the semantic scope of the initial query as little as possible. Based on a predefined vocabulary materialized by fuzzy partitions that linguistically describes the attribute domains, we propose to identify the predicates which are the most correlated to the initial query. Moreover, we consider that the queries involve a user-specified quantitative threshold k corresponding to the approximate number of expected results (the best ones). To assist the user through the reduction of a plethoric answer set to a subset containing approximately k results, we again propose to make use of precomputed fuzzy cardinalities that constitute useful knowledge about the data distributions.

The remainder of the chapter is structured as follows. Section 2 provides a concise reminder about fuzzy sets and fuzzy queries. In Sect. 3, we present the context of our work and especially the fuzzy cardinality-based summarization process. Sections 4 and 5 respectively deal with the two symmetrical problems, i.e. the PAS problem and the explanation of failing queries. We address these issues using a uniform framework based on the notion of fuzzy cardinalities. Experimental results are presented and analyzed in Sect. 6. Section 7 discusses related work, whereas Sect. 8 recalls the main contributions and outlines perspectives for future work.

2 Preliminaries

2.1 Fuzzy Sets

Fuzzy set theory was introduced by Zadeh [22] for modeling classes or sets whose boundaries are not clear-cut. For such objects, the transition between full membership and full mismatch is gradual rather than crisp. Typical examples of such fuzzy classes are those described using adjectives of the natural language, such as *young, cheap, fast*, etc. Formally, a fuzzy set F on a referential U is characterized by a membership function $\mu_F : U \rightarrow [0, 1]$ where $\mu_F(u)$ denotes the grade of membership of u in F. In particular, $\mu_F(u) = 1$ reflects full membership of u in F, while $\mu_F(u) = 0$ expresses absolute non-membership. When $0 < \mu_F(u) < 1$, one speaks of partial membership.

Two crisp sets are of particular interest when defining a fuzzy set F:

- the core $C(F) = \{u \in U \mid \mu_F(u) = 1\}$, which gathers the *prototypes* of F,
- the support $S(F) = \{u \in U \mid \mu_F(u) > 0\}$.

The notion of an α-cut encompasses both these concepts. The α-cut (resp. strict α-cut) F_α (resp. $F_{\overline{\alpha}}$) of a fuzzy set F is defined as the set of elements from the referential which have a degree of membership to F at least equal to (resp. strictly greater than) α:

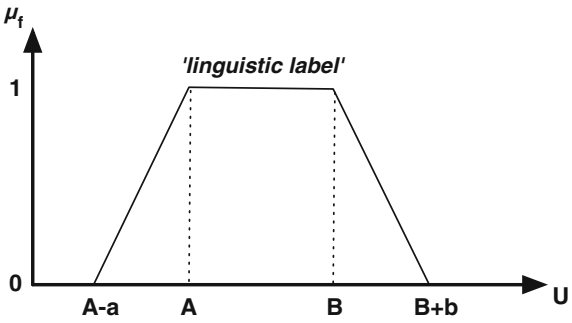

Fig. 1 Trapezoidal membership function

$$F_\alpha = \{u \in U \mid \mu_F(u) \geq \alpha\}$$

$$F_{\overline{\alpha}} = \{u \in U \mid \mu_F(u) > \alpha\}.$$

Straightforwardly, one has: $C(F) = F_1$ and $S(F) = F_{\overline{0}}$.

In practice, the membership function associated with F is often of a trapezoidal shape. Then, F is expressed by the quadruplet (A, B, a, b) where $C(F) = [A, B]$ and $S(F) = [A - a, B + b]$, see Fig. 1.

Let F and G be two fuzzy sets on the universe U, we say that $F \subseteq G$ iff $\mu_F(u) \leq \mu_G(u)$, $\forall u \in U$. The complement of F, denoted by F^c, is defined by $\mu_{F^c}(u) = 1 - \mu_F(u)$. Furthermore, $F \cap G$ (resp. $F \cup G$) is defined the following way: $\mu_{F \cap G}(u) = min(\mu_F(u), \mu_G(u))$ (resp. $\mu_{F \cup G}(u) = max(\mu_F(u), \mu_G(u))$).

As usual, the logical counterparts of the theoretical set operators $\cap$, $\cup$ and the complementation operator correspond respectively to the conjunction $\wedge$, disjunction $\vee$ and negation $\neg$. See [16] for more details.

2.2 Fuzzy Queries and SQLf

Fuzzy sets are convenient tools to model vague criteria and user's preferences. The underlying fuzzy set theory offers a large panoply of connectives to aggregate these preferences following different semantics. Fuzzy sets are used to model and represent common sense properties like '*recent*', '*low*', '*very cheap*', '*large*', ..., that correspond to familiar and easily understandable notions for end users. Moreover, in accordance with the imprecise nature of the concepts they represent, the fuzzy sets behind these properties introduce some graduality when checking the satisfaction of the items wrt. the user's preferences. This gradual satisfaction provides the necessary information to rank order the items that somewhat satisfy the user's requirements.

The language called SQLf described in [8, 29] extends SQL so as to support fuzzy queries. The general principle consists in introducing gradual predicates wherever it makes sense. The three clauses *select*, *from* and *where* of the base block of SQL

are kept in SQLf and the *from* clause remains unchanged. The principal differences affect mainly two aspects :

- the calibration of the result since it is made with discriminated elements, which can be achieved through a number of desired answers (k), a minimal level of satisfaction (α), or both, and
- the nature of the authorized conditions as mentioned previously.

Therefore, the base block is expressed as:
select [**distinct**] [$k \mid \alpha \mid k, \alpha$] *attributes*
from *relations*
where *fuzzy-condition*
where *fuzzy-condition* may involve both Boolean and fuzzy predicates. This expression is interpreted as:

- the fuzzy selection of the Cartesian product of the relations appearing in the *from* clause,
- a projection over the attributes of the *select* clause (duplicates are kept by default, and if *distinct* is specified the maximal degree is attached to the representative in the result),
- the calibration of the result (top k elements and/or those whose score is over the user-specified threshold α_u).

The operations from relational algebra—on which SQLf is based—are extended to fuzzy relations by considering fuzzy relations as fuzzy sets on the one hand and by introducing gradual predicates in the appropriate operations (selections and joins especially) on the other hand. The definitions of these extended relational operators can be found in [4]. As an illustration, we give the definitions of the fuzzy selection and join operators hereafter, where r and s denote two fuzzy relations defined on the sets of attributes X and Y.

- $\mu_{select(r,\, cond)}(t) = \top(\mu_r(t),\ \mu_{cond}(t))$ where $cond$ is a fuzzy predicate and $\top$ is a triangular norm (most usually, *min* is used),
- $\mu_{join(r,\, s,\, A\, \theta\, B)}(tu) = \top(\mu_r(t),\ \mu_s(u),\ \mu_\theta(t.A,\ u.B))$ where A (resp. B) is a subset of X (resp. Y), A and B are defined over the same domains, and θ is a binary relational operator (possibly fuzzy).

A typical example of a fuzzy query is: “retrieve the recent and low-mileage cars”, where *recent* and *low-mileage* are gradual predicates represented by means of fuzzy sets as illustrated in Fig. 2.

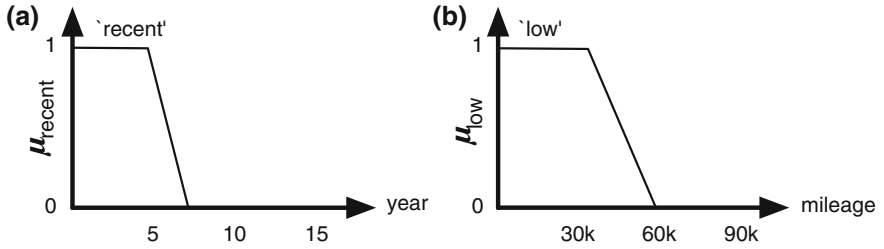

Fig. 2 Predicates: **a** recent and **b** low-mileage

3 Fuzzy-Cardinality-Based Database Summaries

3.1 A Fuzzy-Partition-Based Predefined Vocabulary

Fuzzy sets constitute an interesting framework for extracting knowledge on data that can be easily comprehensible by humans. Indeed, associated with a membership function and a linguistic label, a fuzzy set is a convenient way to formalize a gradual property. As noted in some previous works, especially in [27], such prior knowledge can be used to represent what the authors call a "macro expression of the database". Contrary to the approach presented in [27] where this knowledge is computed by means of a fuzzy classification process, it is, in our approach, defined *a priori* by means of a partition in the sense of Ruspini [33] of each attribute domain. These partitions form a predefined and shared vocabulary and it is assumed that the fuzzy sets involved in users' flexible queries are taken from this vocabulary.

Let R be a relation containing w tuples $\{t_1, t_2, \ldots, t_w\}$ defined on a set Z of q categorical or numerical attributes $\{Z_1, Z_2, \ldots, Z_q\}$. A shared predefined vocabulary on R is defined by means of fuzzy partitions of the q domains. A partition $\mathscr{P}_i$ associated with the domain of attribute Z_i is composed of m_i fuzzy predicates $\{P_{i,1}, P_{i,2}, \ldots, P_{i,m_i}\}$, such that for all Z_i and for all $t \in R$:

$$\sum_{j=1}^{m_i} \mu_{P_{ij}}(t) = 1.$$

As mentioned above, we consider Ruspini partitions for numerical attributes (Fig. 3), i.e., fuzzy partitions composed of fuzzy sets, where a set, say P_i, can only overlap with its predecessor P_{i-1} or/and its successor P_{i+1} (when they exist). For categorical attributes, we simply impose that for each tuple the sum of the satisfaction degrees on all elements of a partition is equal to 1. These partitions are specified by an expert during the database design step and represent "common sense partitions" of the domains. Each $\mathscr{P}_i$ is associated with a set of linguistic labels $\{L^p_{i,1}, L^p_{i,2}, \ldots, L^p_{i,m_i}\}$, each of them corresponding to an adjective which gives the meaning of the fuzzy

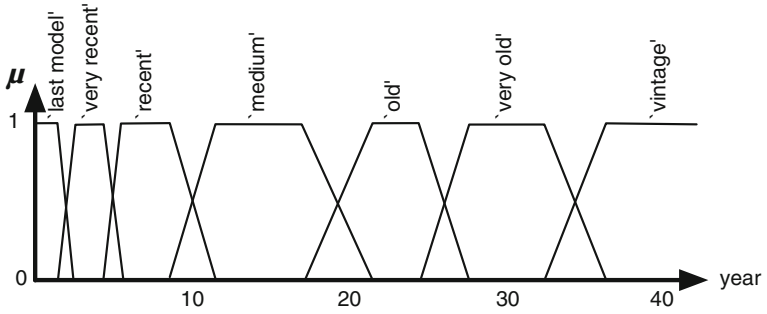

Fig. 3 A partition over the domain of attribute *year*

Table 1 A partition over the domain of attribute *make*

	make															
	dodge	jeep	...	honda	...	nissan	renault	peugeot	dacia	...	ARO	oltcit	...	vw	Lamborghini	Skoda	...
'American'	1	1	...	0	...	0	0	0	0	...	0	0	...	0	0	0	...
'Asian'	0	0	...	1	...	0.6	0	0	0	...	0	0	...	0	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
'French'	0	0	...	0	...	0.4	1	1	0.4	...	0	0	...	0	0	0	...
'East-european'	0	0	...	0	...	0	0	0	0.6	...	1	1	...	0	0	0	...
'German'	0	0	...	0	...	0	0	0	0	...	0	0	...	1	0.5	0.6	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

predicate. A query Q to this relation R is a conjunction of fuzzy predicates chosen among the predefined ones which form the partitions.

As an example, let us consider a database containing ads about second hand cars and a view named *secondHandCars* of schema ($\underline{id}$, *model*, *description*, *year*, *mileage*, *price*, *make*, *length*, *height*, *nbseats*, *consumption*, *acceleration*, *co2emission*) as the result of a join-query over the database. A common sense partition and labelling of the domain of attribute *year* is illustrated in Fig. 3. Table 1 shows a possible common sense partition and labelling of the domain of the categorical attribute *make*.

3.2 About Fuzzy Cardinalities and Their Computation

Hereafter, we describe a technique aimed at building fuzzy database summaries that can be helpful in a cooperative answering perspective.

In the context of flexible querying, fuzzy cardinalities appear to be a convenient formalism to represent how many tuples from a relation satisfy a fuzzy predicate to various degrees. We assume in the following that these various membership degrees are defined by a finite scale $1 = \sigma_1 > \sigma_2 > \cdots > \sigma_f > 0$. Such fuzzy cardinalities can be incrementally computed and maintained for each linguistic label and for the diverse conjunctive combinations of these labels. Fuzzy cardinalities are represented

by means of a possibility distribution [15] like

$$F_{P^a} = 1/0 + \cdots 1/(n-1) + 1/n + \lambda_1/(n+1) + \cdots + \lambda_k/(n+k) + 0/(n+k+1) + \cdots,$$

where $1 > \lambda_1 \geq \cdots \geq \lambda_k > \lambda_{k+1} = 0$ for a predicate P^a. This expression represents a cardinality that possibly equals at least n to degree 1 and possibly equals at least $(n+k)$ to degree λ_k. In this chapter, without loss of information, we use a more compact representation:

$$F_{P^a} = \sigma_1/c_1 + \sigma_2/c_2 + \cdots + \sigma_f/c_f,$$

where $c_i, i = 1..f$ is the number of tuples in the concerned relation that are P^a with a degree at least equal to σ_i. For the computation of cardinalities concerning a conjunction of q fuzzy predicates, like $F_{P^a \wedge P^b \wedge \cdots \wedge P^q}$, one takes into account the minimal satisfaction degree obtained by each tuple t for the concerned predicates, $min(\mu_{P^a}(t), \mu_{P^b}(t), \ldots, \mu_{P^q}(t))$.

As illustrated by Algorithm 1, the computation of the fuzzy cardinalities relies on a single scan of the database but for each tuple, one has to compute its satisfaction degree regarding every possible conjunction of the fuzzy predicates involved in the query. The number of all possible conjunctions to consider is equal to 2^q where q is the number of predicates in the query, but the computation has a linear data complexity and the process remains tractable as soon as q is reasonably small, which is the case in practice (in general, $q \leq 10$). Indeed, even though databases are getting larger and larger, the number of predicates involved in users queries remains stable around half a dozen of predicates. Section 6 illustrates this observation in a concrete applicative context.

The computation of fuzzy cardinalities relies on two steps. First a lattice is generated to store the fuzzy cardinalities according to all the possible conjunctions of predicates. Figure 4 shows the lattice generated for a set of three predicates $\{P^a, P^b, P^c\}$.

Then, for each tuple t from the concerned database $\mathscr{D}$, one computes its performance vector $\langle \mu_{P^a}(t), \mu_{P^b}(t), \ldots, \mu_{P^q}(t) \rangle$ that stores the satisfaction degrees of t

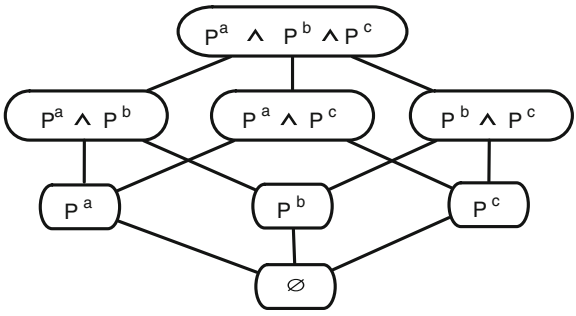

Fig. 4 Lattice of possible conjunctions for a set of three predicates $\{P^a, P^b, P^c\}$

wrt. the atomic predicates $\{P^a, P^b, \ldots, P^q\}$ involved in the query. Using a depth or breadth first exploration of the lattice, one updates the fuzzy cardinalities according to the performance vector of the tuple. Thus, according to the currently analyzed predicate or conjunction of predicates P and a tuple t, if t satisfies P with a degree greater or equal than α_u then the function *updateCardinality* simply increments the stored fuzzy cardinality of P for each $\alpha \geq \alpha_u$. A t-norm, here the minimum, is used by the function *computeSatisDegree* to compute the satisfaction degree of a tuple according to a conjunction of predicates. This satisfaction degree is directly computed on the performance vector associated with the currently analyzed tuple (line 1.6). During the exploration of the lattice, for a tuple t, a path is discarded as soon as t does not satisfy at all the query, say Q, composed of the current conjunction of predicates since

$$\mu_Q(t) = 0 \Rightarrow \forall Q' \text{ such that } pred(Q) \subseteq pred(Q'), \mu_{Q'}(t) = 0$$

where $pred(Q)$ denotes the set of predicates involved in Q.

Input: a failing query $Q = P_a \wedge \ldots \wedge P_n$; a scale of degrees $A = \alpha_f < ... < \alpha_2 < (\alpha_1 = 1)$; a user-defined qualitative threshold α_u;
Output: L a lattice of fuzzy cardinalitites for Q;

```
1.1 begin
1.2   R ← execute(P_a ∨ ... ∨ P_n);
1.3   L ← generateLattice({P_a, ..., P_n});
1.4   //L points to the entry node (∅) of the lattice
1.5   foreach t ∈ R do
1.6     ⟨μ_Pa(t), μ_Pb(t), ..., μ_Pq(t)⟩ ← computePerfVector(t, {P_a, ..., P_n});
1.7     updateLattice(L, ⟨μ_Pa(t), μ_Pb(t), ..., μ_Pq(t)⟩);
1.8     end
1.9 end
```

Algorithm 1: Fuzzy Cardinalities Computation

Two strategies can be envisaged to compute the fuzzy cardinalities: dynamically or *a priori*. The dynamic computation of fuzzy cardinalities allows for the use of user defined fuzzy predicates inside queries. However, to perform an efficient dynamic computation of the fuzzy cardinalities, it would be necessary to modify the optimizer of the DBMS so as to integrate this process in their execution plan.

Input: a lattice of fuzzy cardinalities L; a performance vector $V = \langle \mu_{P^a}(t), \mu_{P^b}(t), ..., \mu_{P^q}(t) \rangle$; a scale of degrees $A = \alpha_f < ... < \alpha_2 < (\alpha_1 = 1)$; a user-defined qualitative threshold α_u;

```
2.1  begin
2.2      N ← Parent(L);
2.3      foreach node ∈ N do
2.4          let P be the predicate associated with the node N;
2.5          μ_P(t) = computeSatisDegree(V, P);
2.6          if μ_P(t) ≥ α_u then
2.7              updateCardinality(N, μ_P(t));
2.8              updateLattice(N, V, A, α_u);
2.9          end
2.10     end
2.11 end
```

Algorithm 2: Recursive function *updateLattice*

In the following, we assume that a shared vocabulary is *a priori* defined by means of fuzzy partitions over the domain of each searchable attribute. Thus, fuzzy cardinalities can be pre-computed for each possible conjunction of predicates taken from the shared vocabulary. More precisely, one computes the fuzzy cardinalities for all the possible conjunctions of predicates containing no more than one predicate of each attribute partition. Indeed, we consider that it does not make sense to explore conjunctions of predicates from the same attribute partition like '*year is young and year is old*'. Fuzzy cardinalities associated with conjunctions which are somewhat satisfied by at least one tuple are stored in a dedicated table of the database. This table can be easily maintained as the fuzzy cardinalities can be updated incrementally [5].

An index computed on the string representation of each conjunction makes it possible to efficiently access the different fuzzy cardinalities.

3.3 A Semantic Correlation Measure

In this subsection, we introduce a measure aimed at assessing the extent to which two fuzzy predicates are semantically correlated. This measure will be used in the approach presented in Sect. 5 as a basis to the augmentation of a query leading to a plethoric answer set.

Given two predicates P^a and P^b, an association rule denoted by $P^a \Rightarrow P^b$ expresses that tuples which are P^a are also P^b (P^a and P^b can be replaced by any conjunction of predicates). As suggested in [9], the confidence of such an association may be quantified by means of a scalar or by a fuzzy (relative) cardinality. The first representation (as a scalar) is used in our approach as it appears more convenient and easier to interpret. Thus, the confidence of an association rule $P^a \Rightarrow P^b$, denoted

by $conf(P^a \Rightarrow P^b)$, is computed as follows:

$$conf(P^a \Rightarrow P^b) = \frac{\Gamma_{P^a \wedge P^b}}{\Gamma_{P^b}}. \quad (1)$$

Here, $\Gamma_{P^a \wedge P^b}$ and Γ_{P^a} correspond to scalar cardinalities, which are computed as the weighted sum of the elements belonging to the associated fuzzy cardinalities. For example, the scalar version of $\Gamma_{recent} = 1/6 + 0.6/7 + 0.2/8$ is $\Gamma_{recent} = 1 \times 6 + 0.6 \times (7 - 6) + 0.2 \times (8 - 7) = 6.8$.

To quantify the semantic link between a query Q and a predicate P, one computes a correlation degree denoted by $\mu_{cor}(P,\ Q)$, as:

$$\mu_{cor}(P,\ Q) = \top(conf(Q \Rightarrow P),\ conf(P \Rightarrow Q)) \quad (2)$$

where $\top$ stands for a t-norm and the minimum is used in our experimentation (Sect. 6.3). One can easily check that this correlation degree is both reflexive $\mu_{cor}(Q,\ Q) = 1$ and symmetric $\mu_{cor}(P,\ Q) = \mu_{cor}(Q,\ P)$.

4 Query Failure Explanation

In this section, we show how fuzzy cardinalities can be used inside a cooperative system to explain failing queries. We still assume that the queries are composed of fuzzy predicates chosen in a predefined vocabulary. Let us first recall the formal definition of the EAS problem.

Definition 1 Let Q be a fuzzy query and $\Sigma_Q = \{t \in D \mid \mu_Q(t) > 0\}$ the set of answers to Q against a given database D. We say that Q results in the EAS problem if $\Sigma_Q = \emptyset$.

4.1 About Minimal Failing and Unsatisfactory Subqueries

An empty set of answers associated with a fuzzy query $Q = P_1 \wedge P_2 \wedge \cdots \wedge P_n$ is necessarily due to an empty support (w.r.t. the current state of the database) for at least one of the *subqueries* of Q. The notion of an unsatisfactory set of answers generalizes this problem by considering an empty α-cut of Q where α is a user-defined qualitative threshold. As explained in Sect. 2.1, the support and the core of a fuzzy set are particular cases of α-cuts where α is respectively equal to 0^+ and 1. In the rest of the chapter we only use the notion of an empty α-cut to refer to failing queries as well as unsatisfactory ones.

Thus, an extreme case of a failing query corresponds to an empty 1-cut for Q only. The opposite extreme is when one or several predicates P_i have an empty 0^+-cut.

Between these two situations, it is of interest to detect the subqueries composed of more than one predicate and less than n predicates, which have an empty 0^+-cut. From an empty to an unsatisfactory set of answers, the problem defined above just has to be slightly revisited, where the condition of an empty 0^+-cut is transposed to α-cuts, where α is taken from a predefined scale of membership degrees $\mathscr{S} : 1 = \alpha_1 > \alpha_2 > \cdots > \alpha_f = 0^+$.

Definition 2 Let us consider a query $Q = P_1 \wedge P_2 \wedge \cdots \wedge P_n$, and let S and S' be two subsets of predicates such that $S' \subset S \subseteq \{P_1, P_2, \ldots, P_n\}$. A conjunction of elements from S (resp. S') is a *subquery* (resp. *strict subquery*) of Q.

If one wants to explain why the result of the initial query is empty (resp. unsatisfactory), and/or weaken the query by identifying the subqueries whose α-cut is empty, one must naturally require that such subqueries be minimal: a subquery Q' of a query Q constitutes a minimal explanation if the considered α-cut is empty and if no (strict) subquery of Q' has an empty α-cut. This corresponds to a generalization of the concept of a *Minimal Failing Subquery* (MFS) [18].

Let us denote by Σ_Q^α the set of answers to the α-cut of a query Q against a given database D: $\Sigma_Q^\alpha = \{t \in D \mid \mu_Q(t) \geq \alpha\}$.

Definition 3 A *Minimal Failing Subquery* of a query $Q = P_1 \wedge P_2 \wedge \cdots \wedge P_n$ for a given α is any subquery Q' of Q such that $\Sigma_{Q'}^\alpha = \emptyset$ and for all strict subquery Q'' of Q', $\Sigma_{Q''}^\alpha \neq \emptyset$.

When faced with an empty set of answers for a user-defined threshold α, the explanation process that we propose in this chapter generates layered MFSs for different satisfaction degrees $\alpha_i \in \mathscr{S}$, $\alpha_i \in [\alpha, 1]$.

Obviously, due to the monotonicity of inclusion of α-cuts, one has $\Sigma_Q^{\alpha_i} \subseteq \Sigma_Q^{\alpha_j}$ if $\alpha_i \geq \alpha_j$. Therefore, a query Q that fails for a given α_j also fails for higher satisfaction degrees $\alpha_i > \alpha_j$. However, this property is not satisfied by minimal failing subqueries. Indeed, a subquery Q' can be an MFS of Q for a given α_j without being minimal for higher satisfaction degrees $\alpha_i > \alpha_j$ as a strict subquery of Q', say Q'', may fail for α_i and not for α_j.

During the layered MFS detection step (Sect. 4.2), when a subquery Q' of an initial failing or unsatisfactory query Q is detected for a degree α_j, one has to check for each higher level $\alpha_i > \alpha_j$ if Q' is also minimal at the level α_i before considering Q' as an MFS for this level.

4.2 Cardinality-Based MFS Detection

Using the precomputed fuzzy cardinalities, one can detect the MFSs for different empty α-cuts of Q, starting from a user-defined qualitative threshold up to the highest satisfaction degree 1.

In the manner of Apriori [1], Algorithm 3 starts with atomic predicates and the first α_i-cut of interest, the one corresponding to the user-defined qualitative threshold α_u. To determine if an atomic predicate P_a is a failing subquery of Q, one just has to check the associated precomputed fuzzy cardinality. If no tuple satisfies P_a at least with the degree α_i then P_a, as an atomic predicate, is by definition an MFS of Q and is also an MFS for $\alpha_j > \alpha_i$. Then, for the second round of the loop (line 3.7 of Algorithm 3), conjunctions containing two non failing predicates are generated and for each of them (line 3.11) one checks the fuzzy cardinalities so as to determine if it is an MFS. If one of these conjunctions, say $P_a \wedge P_b \wedge P_c$, is an MFS for a degree α_i one tries to propagate it to higher satisfaction degrees (see Algorithm 4 where $isMFS(L, MFS_{\alpha_j}(Q))$ returns *true* if $L \in MFS_{\alpha_j}(Q)$, *false* otherwise). As the MFS property is not monotonic with respect to α-cuts, one checks with Algorithm 4 for each $\alpha_j > \alpha_i$ if a subquery of $P_a \wedge P_b \wedge P_c$ corresponds to a previously detected MFS for degree α_j; if it is not the case $P_a \wedge P_b \wedge P_c$ is stored as an MFS of Q for α_j. Obviously, an atomic failing query is an MFS for all α-cuts. Then, the algorithm goes back to the main loop (line 3.7) and conjunctions containing three predicates are generated for each considered satisfaction degree (line 3.8) taking care that these conjunctions do not contain an already identified MFS. This recursive process goes on until candidate conjunctions cannot be generated anymore.

The complexity of this algorithm is obviously exponential in the number of predicates involved in the failing query to explain, where the worst case corresponds to a single MFS Q for the maximal satisfaction degree of 1. In this case, the *foreach* loop (line 3.11) makes 2^n iterations where n is the number of predicates in Q. For a complete gradual explanation from $\alpha = 0^+$ to 1, the 2^n iterations are repeated f times, where f is the number of considered satisfaction degrees in $\mathscr{S} : 1 = \alpha_1 > \alpha_2 > \cdots > \alpha_f = 0^+$. Thus, the final complexity in the worst case is $f \times 2^n \in \theta(2^n)$. As we said previously, this is not a problem in practice as the number of predicates specified by a user is rather low (≤ 10) in most applicative contexts. Therefore, this process remains tractable as we will show experimentally in Sect. 6.

Once the MFSs have been detected, it is possible to inform the user about the conflicts in his/her query, which should help him/her revise the selection condition of the failing query.

5 Plethoric Answer Set Reduction

In this section, we address the problem symmetrical to that studied in Sect. 4: the Plethoric Answer Sets (PAS) problem. Let Q be a fuzzy query and Σ_Q (denoted also by $\Sigma_Q^{0^+}$) its set of answers against the database D. One can write $\Sigma_Q^{0^+} = \{\mu_1/t_1, \mu_2/t_2, \ldots, \mu_n/t_n\}$ where t_i is a tuple of D and μ_i its satisfaction degree w.r.t. Q. Assume that a user provides a number k of desired answers along with the query Q.

Input: a failing query $Q = P_1 \wedge \ldots \wedge P_n$;
a scale of degrees $A = 0 < \alpha_f < ... < \alpha_2 < (\alpha_1 = 1)$;
a user-defined qualitative threshold α_u;
Output: $MFS(Q)$ ordered sets of MFS's of Q, one set for each α-cut of Q.

3.1 **begin**
3.2 **foreach** $\alpha_i \in A \mid \alpha_i \geq \alpha_u$ **do**
3.3 $MFS_{\alpha_i}(Q) \leftarrow \emptyset$; $E_{\alpha_i} \leftarrow \{P_1, \ldots, P_n\}$;
3.4 $Cand_{\alpha_i} \leftarrow E_{\alpha_i}$;
3.5 **end**
3.6 $nbPred \leftarrow 1$;
3.7 **while** $Cand_{\alpha_1} \neq \emptyset$ **do**
3.8 **foreach** $\alpha_i \in A \mid \alpha_i \geq \alpha_u$ **do**
3.9 // generation of the candidates of size nbPred
3.10 $Cand_{\alpha_i} \leftarrow \{M$ composed of $nbPred$ predicates present in E_{α_i} such that $\forall M' \subset M,\ M' \notin MFS_{\alpha_i}(Q)\}$;
3.11 **foreach** L *in* $Cand_{\alpha_i}$ **do**
3.12 **if** $card(L_{\alpha_i}) = 0$ **then**
3.13 $MFS_{\alpha_i}(Q) \leftarrow MFS_{\alpha_i}(Q) \cup \{L\}$;
3.14 //E_L contains the atomic predicates that compose L
3.15 $E_{\alpha_i} \leftarrow E_{\alpha_i} - E_L$;
3.16 //Propagate L to higher satisfaction degrees
3.17 // $E = \cup_i\ E_{\alpha_i}$ and $MFS = \cup_i\ MFS_{\alpha_i}(Q)$
3.18 *propagate*(α_i, A, L, MFS, E);
3.19 **end**
3.20 **end**
3.21 **end**
3.22 $nbPred \rightarrow nbPred + 1$;
3.23 **end**
3.24 **end**

Algorithm 3: Gradual MFS computation

Definition 4 We say that a PAS problem occurs for Q if $\left|\Sigma_Q^{\mu_{max(Q)}}\right| \gg k$.

Where $\mu_{max(Q)} = sup_{t_i \in \Sigma_Q^{0^+}} \mu_i$ and $\Sigma_Q^{\mu_{max(Q)}} = \{t_i \in \Sigma_Q^{0^+} \mid \mu_i = \mu_{max(Q)}\}$.

This definition means that the set of answers $\Sigma_Q^{0^+}$ contains a large number of answers (with a maximal satisfaction degree) w.r.t. the number k of desired answers. The general idea of our solution is to augment a user query Q with predefined predicates which are semantically correlated with those present in Q, in order to reduce the initial answer set and get an answer subset whose cardinality is as close to k (the user-specified quantitative threshold) as possible.

Input: a satisfaction degree: α_i; a scale of degrees: A; detected MFS for α_i: L; a reference to the array of layered MFS: MFS; a reference to the array of predicates used for the generation of candidates: E;

```
4.1  procedure propagate(α_i, A, L, MFS, E) begin
4.2      foreach α_j ∈ A | α_j ≥ α_i do
4.3          if isAtomic(L) or isMFS(L,MFS_{α_j}(Q)) then
4.4              MFS_{α_j}(Q) ← MFS_{α_j}(Q) ∪ {L};
4.5              E_{α_j} ← E_{α_j} − E_L;
4.6              else
4.7                  break;
4.8              end
4.9          end
4.10     end
4.11 end
```

Algorithm 4: Procedure that propagates an MFS to higher satisfaction degrees

5.1 Correlation-Based Ranking

In the approach we propose, the new conjuncts to be added to the initial query are chosen among a set of possible predicates pertaining to the attributes of the schema of the database queried (see Sect. 3.1). This choice is mainly made according to their correlation with the initial query. A user query Q is composed of n ($\geq$ 1) specified fuzzy predicates, denoted by $P^{s_1}, P^{s_2}, \ldots, P^{s_n}$, which come from the predefined vocabulary associated with the database (Sect. 3.1). The first step of the query augmentation process is to identify the predefined predicates most correlated to the initial query Q.

The notion of correlation introduced in Sect. 3.3 is used to qualify and quantify the extent to which two fuzzy sets (one associated with a predefined predicate $P^p_{i,j}$, the other associated with the initial query Q) are somewhat "semantically" linked.

Using the fuzzy-cardinality-based measure of correlation (cf. Formula 2), we can identify the predefined predicates most correlated to an under-specified query Q. In practice, we only consider the η most correlated predicates to a query, where η is a technical parameter which has been set to 5 in our experimentation. This limitation is motivated by the fact that an augmentation process involving more than η iterations, i.e., the addition of more than η predicates, could lead to important modifications of the scope of the initial query. Those η predicates most correlated to Q are denoted by $P^{c^1}_Q, P^{c^2}_Q, \ldots, P^{c^\eta}_Q$.

5.2 Reduction-Based Reranking

The second step of the query augmentation process aims at reranking the η predicates most correlated to the query according to their "reduction capability". It is assumed that the user specifies a value for the parameter k which defines the number of answers he/she expects. Let $F_{Q \wedge P_Q^{c^r}}$, $r = 1..\eta$, be the fuzzy cardinality of the answer set when Q is augmented with $P_Q^{c^r}$. $P_Q^{c^r}$ is all the more interesting for augmenting Q as $Q \wedge P_Q^{c^r}$ contains a σ_i-cut ($\sigma_i \in \mathscr{S}$ and $\sigma_i \geq \alpha_u$) with a cardinality c_i close to k and σ_i close to 1. To quantify how interesting $P_Q^{c^r}$ is, we compute for each σ_i-cut of $F_{Q \wedge P_Q^{c^r}}$ a "strengthening degree" which represents a compromise between its membership degree σ_i and its associated cardinality c_i. The global degree assigned to $F_{Q \wedge P_Q^{c^r}}$, denoted by $\mu_{stren}(F_{Q \wedge P_Q^{c^r}})$, is the maximum of its strengthening degrees over the different σ_i-cuts:

$$\mu_{stren}(F_{Q \wedge P_Q^{c^r}}) = sup_{1 \leq i \leq f} \top \left(1 - \frac{|c_i - k|}{max(k,\ |\Sigma_Q| - k)},\ \sigma_i\right)$$

where $\top$ stands for a t-norm and the minimum is used in our experimentation. This reranking of the predicates the most correlated to Q can be carried out using the fuzzy cardinalities associated with each conjunction $Q \wedge P_Q^{c^r}$, $r = 1..\eta$.

Example 1 To illustrate this reranking strategy, let us consider a user query Q resulting in a PAS problem ($|\Sigma_Q^*| = 123$ and $|\Sigma_Q| = 412$), where k has been set to 50. As an example, let us consider the following candidates $P_Q^{c^1}$, $P_Q^{c^2}$, $P_Q^{c^3}$, $P_Q^{c^4}$, $P_Q^{c^5}$ and the respective fuzzy cardinalities:

- $F_{Q \wedge P_Q^{c^1}} = \{1/72 + 0.8/74 + 0.6/91 + 0.4/92 + 0.2/121\}$, $\mu_{stren}(F_{Q \wedge P_Q^{c^1}}) \simeq 0,94$
- $F_{Q \wedge P_Q^{c^2}} = \{1/89 + 0.8/101 + 0.6/135 + 0.4/165 + 0.2/169\}$, $\mu_{stren}(F_{Q \wedge P_Q^{c^2}}) \simeq 0,9$
- $F_{Q \wedge P_Q^{c^3}} = \{1/24 + 0.8/32 + 0.6/39 + 0.4/50 + 0.2/101\}$, $\mu_{stren}(F_{Q \wedge P_Q^{c^3}}) \simeq 0,93$
- $F_{Q \wedge P_Q^{c^4}} = \{1/37 + 0.8/51 + 0.6/80 + 0.4/94 + 0.2/221\}$, $\mu_{stren}(F_{Q \wedge P_Q^{c^4}}) \simeq 0,96$
- $F_{Q \wedge P_Q^{c^5}} = \{1/54 + 0.8/61 + 0.6/88 + 0.4/129 + 0.2/137\}$, $\mu_{stren}(F_{Q \wedge P_Q^{c^5}}) \simeq 0,99$.

According to the problem definition ($k = 50$) and the fuzzy cardinalities above, the following ranking is suggested to the user: 1) $P_Q^{c^5}$, 2) $P_Q^{c^4}$, 3) $P_Q^{c^1}$, 4) $P_Q^{c^3}$, 5) $P_Q^{c^2}$. Of course, to make this ranking more intelligible to the user, the candidates are proposed with their associated linguistic labels (cf. the concrete example about used cars given in Sect. 6.3).◇

5.3 Query Augmentation Process

Precomputed Knowledge

As the predicates specified by the user and those that we propose to add to the initial query are chosen among the predefined vocabulary, one can precompute some useful knowledge that will make the augmentation process faster. We propose to compute and maintain precomputed knowledge which is stored in two tables. The first one contains the precomputed fuzzy cardinalities introduced in Sect. 3.2, whereas the second one stores the correlation degrees between sets of predefined predicates (corresponding to the initial query) and any other predefined predicate. Using these correlation degrees, one can also determine and store, for each conjunction of predefined predicates, the most correlated predefined atomic predicates ranked in decreasing order of their correlation degrees. Both tables have to be updated after each (batch of) modification(s) performed on the data but these updates imply a simple incremental computation.

Interactive Augmentation Mechanism

The query augmentation process consists of the following steps. One first checks the table of fuzzy cardinalities in order to determine whether the user is faced with a PAS problem according to the value he/she has assigned to k. If so, one retrieves—still in constant time—up to η candidates that are then reranked according to k and presented to the user. Finally, as it is illustrated in Sect. 6.3, the user can decide to process the initial query, to process one of the suggested augmented queries, or to ask for another augmentation iteration of one of the augmented queries.

6 Experimentation

6.1 Context

The fuzzy-cardinality-based summarization process as well as the cooperative approaches described in Sects. 4 and 5 have been tested on a concrete database containing ads about second hand cars. This database is composed of a single relation named *secondHandCars* and contains 46,089 tuples with the following schema: {*idads*, *year*, *mileage*, *price*, *make*, *length*, *height*, *nbseats*, *acceleration*, *consumption*, *co2emission*}.

Common sense fuzzy partitions have been defined on the attributes of this relation, which led to a shared vocabulary made of 59 fuzzy predicates. Figure 5 illustrates the way users may employ this vocabulary to construct their fuzzy queries.

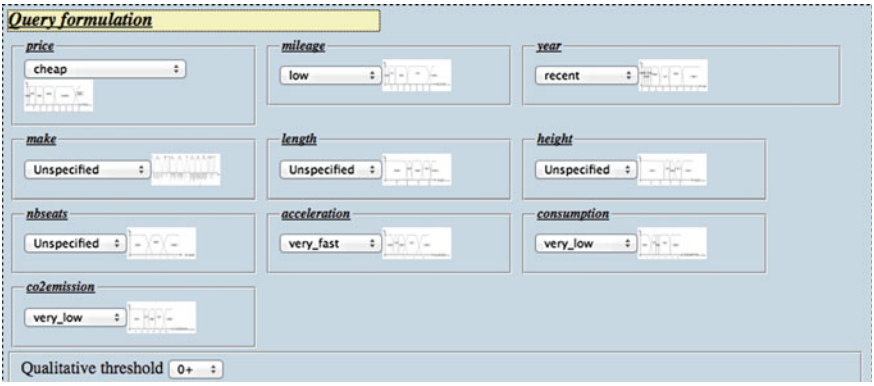

Fig. 5 Query interface relying on the shared vocabulary

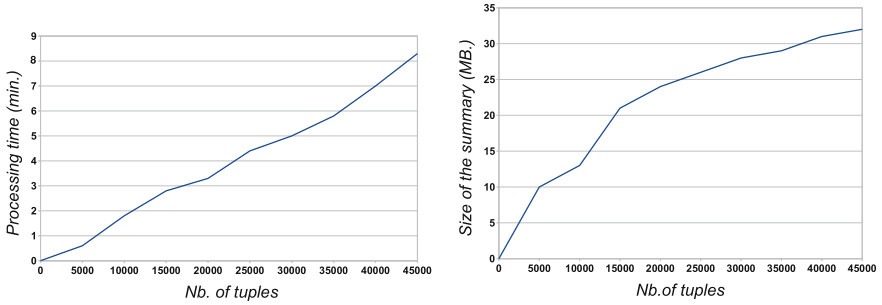

Fig. 6 Evolution of the processing time and space wrt. the number of tuples

Using this predefined vocabulary, we have first evaluated the time needed to compute a complete fuzzy-cardinality-based summary and also its evolution with respect to the size of the database. Figure 6 (left) shows the evolution of the time needed to compute the fuzzy cardinalities for a database whose size varies from 5000 to 45,000. Figure 6 (right) shows the evolution of the memory space needed to store the computed fuzzy-cardinality-based summary. These results have been obtained on a basic computer configuration (Intel Core 2 Duo 2.53GHz with 4Go 1067 MHz of DDR3 ram) and Postgresql as the RDBMS for the storage of the relation *secondHandCars* and its summary.

As expected, the time needed to compute the fuzzy cardinality-based summary linearly increases wrt. the size of the database. The most interesting phenomenon that can be observed in Fig. 6 is that the size of the memory used to store the fuzzy cardinalities is very reasonable and increases in a logarithmic way according to the number of tuples. Indeed, the number of fuzzy cardinalities that have to be stored increases quickly from 0 to 15,000 tuples, then very slowly to 35,000 and is almost stable from 35,000 to 45,000. This phenomenon was predictable and can be explained by the fact that whatever the number of tuples, the possible combinations of

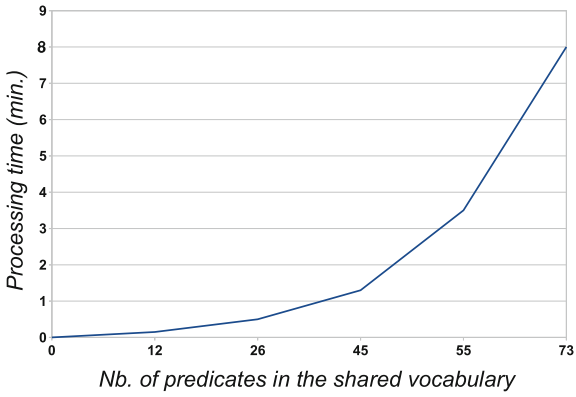

Fig. 7 Evolution of the processing time wrt. the number of predicates

properties to describe them is finite and can quickly be enumerated. As an example, let us consider the failing fuzzy query Q composed of 8 predicates: "*year is recent and mileage is low and price is low and acceleration is very_high and consumption is very_low and co2emission is very_low*". Whatever the number of tuples in the database, some combinations of properties are not observed, such as: "*year is recent and mileage is low and price is low*", "*acceleration is very_high and consumption is very_low*", "*acceleration is very_high and co2emission is very_low*",etc. So, one can expect that the size of the memory used to store the fuzzy cardinalities will not increase significantly in general, even when the database grows a lot.

To complete these observations, Fig. 7 shows the evolution of the time needed to compute a complete summary of the database (with 46.089 tuples) with respect to the number of predicates in the vocabulary.

This first experimentation clearly shows that this fuzzy-cardinality-based summary can be considered even for large databases as long as the vocabulary contains a reasonably small number of fuzzy predicates. It is worth noticing that this characteristics correspond to most of the applicative contexts, especially for web sites proposing a query interface to their database.

6.2 A Prototype for Explaining Failing Queries

The query interface illustrated in Fig. 5 has been completed with the cooperative approach described in Sect. 4 in order to provide the users with some explanations about the failure of their queries [30]. In the first part of this experimentation, we have used the fuzzy cardinalities precomputed according to the predefined vocabulary and estimated the time needed to generate the explanations of failing queries. For this purpose, we have submitted 50 failing or unsatisfactory queries containing various numbers of predicates, from 1 up to 10. Figure 8 illustrates the explanations of the failing query "*year is vintage and price is low*", whereas Fig. 9 shows the evolution

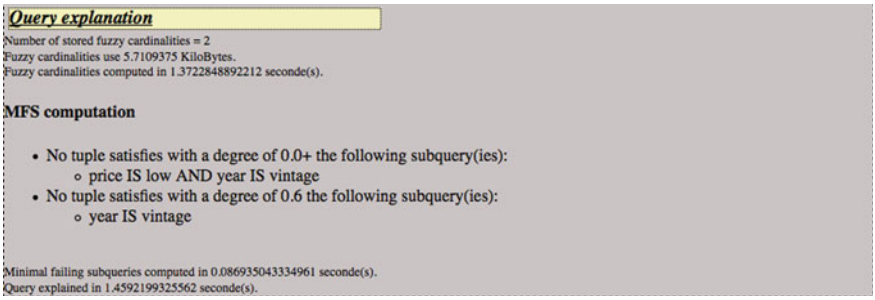

Fig. 8 Explanations for the failing query "*year is vintage and price is low*"

of the average time needed to compute these explanations for failing queries whose number of predicates varies from 1 to 10. These results have been observed on three queries containing a single predicate, six containing two predicates and ten for other numbers of predicates. Despite the exponential aspect of the curve, these results show that for a reasonable number of predicates involved in the query, the time needed to compute the MFSs is very limited. Moreover, it is worth noticing that the performances of this explanation process could certainly be improved using parallel programming and a compiled language such as C instead of PHP.

To complete this first experimentation, we have also implemented the "naive" approach studied in [18], which does not make use of a summary but processes every possible subquery. To make the comparison meaningful, we have implemented a version of our approach where a fuzzy-cardinality-based summary is dynamically computed for each submitted query. In this case, the sole predicates involved in the query are concerned by the summarization process. Figure 10 graphically shows the difference in computation time for these two approaches and empirically shows the benefits of a single scan of the database. This comparison is performed for queries with at most six predicates, as the time needed to compute the MFSs for longer queries is prohibitive with the technique proposed in [18].

In Sect. 6, we have seen that the size of a fuzzy-cardinality-based summary is very limited, and that it is not linearly related to the size of the database but rather depends on the applicative context and on the correlations between the attributes. Indeed, Fig. 6 shows that the size of the summary quickly converges as soon as all the "plausible combinations" of predicates have been enumerated.

The experimentations that we carried out show the benefits of an approach whose complexity is not very sensitive to the number of tuples in the database. However, such an approach can only be used when the queries involve a relatively small number of predicates. As said previously, this is not a problem in practice as the number of predicates specified by a user is rather low (≤ 10) in most applicative contexts. To

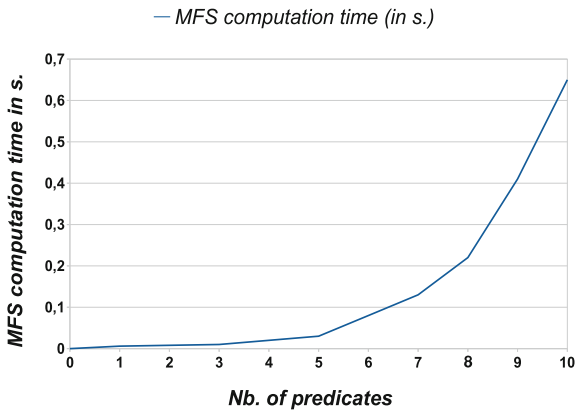

Fig. 9 MFS computation time using precomputed fuzzy cardinalities

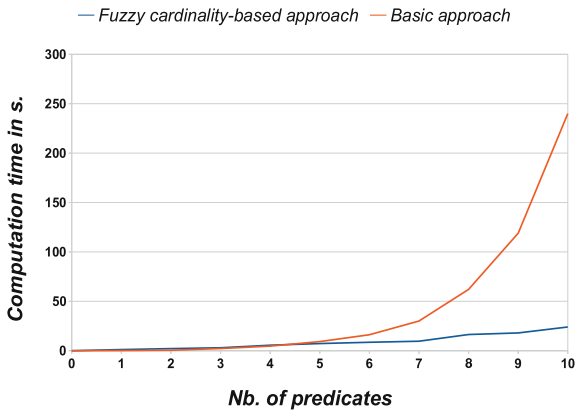

Fig. 10 Naive method versus dynamic computation of the summary

support this assertion, we have analyzed the query interface of 12 web sites[1] proposing an access to ads about second hand cars. The maximum number of constraints (*i.e.* predicates) a user can specify through these interfaces varies from 5 to 12 with an average of 8.8 predicates.

MFSs-Based Failing Queries Revision

When faced with a failing query, the explanations given by the layered MFSs help the user revise his/her initial query. Depending on the nature of the conflicts underlined in the MFSs, a user may:

- reconsider the qualitative threshold α_u specified in the query,

[1] Examples of web portals to databases containing ads about second hand cars: http://www.annoncesauto.com, http://www.paruvendu.fr, http://www.auroreflex.com, http://www.ebay.fr, http://www.lacentrale.fr, ...

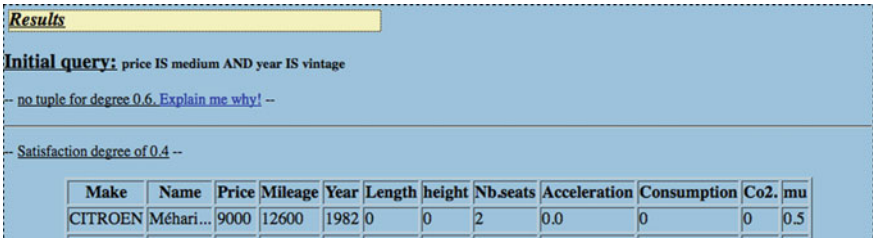

Fig. 11 Example of an MFS-guided revision of an initial failing query

- remove one or several predicates involved in a conflict,
- replace one or several predicates involved in an MFS by predicates from the shared vocabulary that appear less conflicting,
- apply a repair step which aims at relaxing the definition of some predicates [7] or replace the conjunctive query Q by a fuzzy quantified statement of the type $Q^* = most(P_1, P_2, \ldots, P_n)$ [37].

Figure 8 illustrates a failing situation for an initial query "*year is vintage and price is low*" and a user-defined qualitative threshold $\alpha_u = 0.2$. The explanation related to this failure clearly point out that the predicate "*price is low*" is in conflict with the property "*year is vintage*". Guided with this explanation, one may replace the conflicting predicate "*price is low*" by a less demanding one such as "*price is medium*" (Fig. 11).

Thanks to the gradual MFSs, the user knows that it is useless to expect answers with a high level of satisfaction if he/she keeps the predicate "*year is vintage*" which constitutes an atomic MFS for $\alpha = 0.6$.

6.3 A Prototype for Reducing Plethoric Answer Sets

To help users revise their queries when they return a plethoric answer set, we have augmented the query interface illustrated in Fig. 5 with a cooperative functionality that implements the approach described in Sect. 5 [10]. Using this interface, users may define their fuzzy queries and specify a quantitative threshold k corresponding to the number of answers they expect.

A concrete example given below illustrates the relevance of the predicates suggested by the system for augmenting the initial query.

Example 2 Let us consider the following query Q composed of fuzzy predicates chosen among the shared vocabulary (Fig. 5):

$$Q = \textbf{select} * \textbf{from } second\ Hand\ Cars \textbf{ where } year\ is\ very\_old \textbf{ with } k = 50.$$

Executed on the second hand cars DB, Q returns an answer set whose cardinality is: $F_Q = \{1/179 + 0.8/179 + 0.6/179 + 0.4/323 + 0.2/323\}$. We are faced with a PAS problem, which means that the query augmentation process is triggered.

The following candidates are suggested along with the fuzzy cardinality of the corresponding augmented queries:

1. *mileage is medium* ($\mu_{cor}(Q, P^p_{mileage,\, medium}) = 0.11$)

$$F_{Q \wedge P^p_{mileage,\, medium}} = \{1/24 + 0.8/27 + 0.6/28 + 0.4/72 + 0.2/77\}$$

2. *mileage is very high* ($\mu_{cor}(Q, P^p_{mileage,\, veryhigh}) = 0.19$)

$$F_{Q \wedge P^p_{mileage,\, very\, high}} = \{1/7 + 0.8/7 + 0.6/8 + 0.4/18 + 0.2/19\}$$

3. *mileage is high* ($\mu_{cor}(Q, P^p_{mileage,\, high}) = 0.37$)

$$F_{Q \wedge P^p_{mileage,\, high}} = \{1/101 + 0.8/106 + 0.6/110 + 0.4/215 + 0.2/223\}.$$

For each candidate query Q', the user may decide to process Q' (i.e. retrieve the results) or to repeat the augmentation process on Q'. If the latter option is chosen, the table of fuzzy cardinalities is checked in order to retrieve relevant predicates for augmenting Q' (i.e. properties correlated to Q') along with their associated fuzzy cardinalities that are ranked according to k. Let us assume that the user selects

$$Q' = year\ is\ very\_old\ and\ mileage\ is\ medium$$

for a second augmentation step. The following candidates are suggested along with their fuzzy cardinalities:

1. *price is low* ($\mu_{cor}(Q', P^p_{price,\, low}) = 0.34$)

$$F_{Q' \wedge P^p_{price,\, low}} = \{1/18 + 0.8/20 + 0.6/21 + 0.4/46 + 0.2/51\}$$

2. *price is medium* ($\mu_{cor}(Q', P^p_{price,\, medium}) = 0.15$)

$$F_{Q' \wedge P^p_{price,\, medium}} = \{1/6 + 0.8/7 + 0.6/7 + 0.4/22 + 0.2/22\}.\diamond$$

From this experimentation on a real-world database, one may observe that query augmentation based on semantic correlation provides the users with useful information about data distributions and the possible queries that can be formulated in order to retrieve coherent answer sets. By coherent answer set, we mean a group of items that share correlated properties and that may correspond to what the user was looking for without knowing initially how to retrieve them. Moreover, thanks to the precomputed knowledge tables, it is not necessary to process the candidate queries

to inform the user about the size of their answer sets and the predicates that can be used to augment them.

This experimentation shows that the predicates suggested to augment the queries are meaningful and coherent according to the initial underspecified queries. One can find below some examples of suggested augmented queries Q' starting from underspecified queries Q:

- Q = *year is old and mileage is high and price is very_low*
  intensified after two iterations into:
  Q' = *year is old and mileage is high and price is very_low and acceleration is slow and consumption is high*
  with $|\Sigma^*_Q| = 63$ and $|\Sigma^*_{Q'}| = 26$.
- Q = *year is recent*
  intensified after two iterations into:
  Q' = *year is recent and mileage is low and price is medium*
  with $|\Sigma^*_Q| = 4.060$ and $|\Sigma^*_{Q'}| = 199$.
- Q = *price is high*
  intensified after two iterations into:
  Q' = *price is high and year is last_model and co2emission is low*
  with $|\Sigma^*_Q| = 180$ and $|\Sigma^*_{Q'}| = 45$.

7 Related Work

The practical need for endowing intelligent information systems with the ability to exhibit cooperative behavior has been recognized since the early '90s. As pointed out in [13, 17], the main intent of cooperative systems is to provide correct, non-misleading and useful answers, rather than literal answers to user queries. During the last two decades, several cooperative approaches have been proposed for different aspects related to the problems dealt with here. In this section, we first recall the main existing approaches for database summarization. Then, we situate the uniform fuzzy-cardinality-based cooperative approach we propose with respect to work related to failing queries and plethoric answers respectively.

Database Summarization

In [34], Saint-Paul et al. propose an approach to the production of linguistic summaries structured in a hierarchy, i.e., a summarization tree where the tuples from the database are rewritten using the linguistic variables involved in fuzzy partitions of the attribute domains. The deeper the summary in the tree, the finer its granularity. First, the tuples from the database are rewritten using the linguistic variables involved in fuzzy partitions of the attribute domains. Then, each candidate tuple is incorporated into the summarization tree and reaches a leaf node (which can be seen

as a classification of the tuple). In the hierarchical structure, a level is associated with the relative proportion of data that is described by the associated summary. However, the relative semantic poorness of these summaries in terms of cardinality-related information makes its interest limited when it comes to helping the user reformulate his/her query in an EAS or PAS situation.

Developed by Rasmussen and Yager, SummarySQL [32] is a fuzzy query language which can evaluate the truth degree of a summary guessed by the user. A summary expresses knowledge about the database in a statement under the form "Q objects in DB are S" or "Q R objects in DB are S" where DB stands for the database, Q is a linguistic quantifier (*almost all*, *about half*, etc.) and R and S are linguistic terms (*young*, *well-paid*, and so on). The expression is evaluated for each tuple and the associated truth values are later used to obtain a truth value for the whole summary. A similar type of approach is proposed in [28]. Anyway, this view of database summarization is purely oriented toward knowledge discovery, and does not aim at providing tools to support database querying/browsing.

Failing Queries

We discuss here only some studies that are most related to the approach proposed. For a complete and rich synthesis of works about failing queries, the reader can refer to [6, 7]. Jannach [19] proposes an algorithm which is somewhat similar to ours, but which does not precompute the cardinalities. Instead, it builds a binary matrix containing the satisfaction degrees obtained by each tuple for each atomic predicate, and combines these degrees in order to detect the MFSs. The main problems with this technique are that (i) such a table can be very large to the point of not fitting in memory (cf. the experimental results reported in [30]), and (ii) a query is processed for each atomic predicate on the whole dataset.

The algorithm proposed in [18] *processes* every query corresponding to a candidate MFS, which is obviously quite expensive. Similarly, the approach described in [23, 24], processes every maximally successful subquery of a failing query in order to retrieve what the author calls a *recovery set*. Compared to these works, the major interest of our approach is that the determination of the MFSs does not imply any additional query processing, thanks to the precomputation of fuzzy cardinalities. Thus, the complexity of our algorithm is linear in the size of the data (cf. Sect. 6.2).

Finally, apart from the study done in [7] and to the best of our knowledge, there is no other work that has addressed the problem of MFS detection in the context of preference queries, which covers an application context that goes beyond failing queries *stricto sensu*.

Plethoric Answer Sets

In their probabilistic ranking model, Chaudhuri et al. [11] also propose to use a correlation property between attributes and to take it into account when computing

ranking scores. However, correlation links are identified between attributes and not predicates, and the identification of these correlations relies on a workload of past submitted queries.

Su et al. [35] have emphasized the difficulty to manage such a workload of previously submitted queries or users feedbacks. This is why they have proposed to learn attribute importances regarding a *price* attribute and to rank retrieved items according to their commercial interest. Nevertheless, this method is domain-dependent and can only be applied for e-commerce databases.

The approach advocated by Ozawa et al. [26, 27] is also based on the analysis of the database itself, and aims at providing the user with information about the data distributions and the most efficient constraints to add to the initial query in order to reduce the initial set of answers. The approach we propose in this chapter is somewhat close to that introduced in [26], but instead of suggesting an attribute on which the user should specify a new constraint, our method directly suggests a set of fuzzy predicates along with some information about their relative interest with respect to the user needs. The main limitation of the approach advocated in [26] is that the attribute chosen is the one which maximizes the dispersion of the initial set of answers, whereas most of the time, it does not have any semantic link with the predicates that the user specified in his/her initial query. To illustrate this, let us consider again the relation *secondHandCars* introduced in Sect. 3.1. Let Q be a fuzzy query on *secondHandCars*: *"select * from secondhandcars where type = 'estate' and year is recent"* resulting in a PAS problem. In such a situation, Ozawa et al. [26] first apply a fuzzy c-means algorithm [2] to classify the data, and each fuzzy cluster is associated with a predefined linguistic label. After having attributed a weight to each cluster according to its representativity of the initial set of answers, a global dispersion degree is computed for each attribute. The user is then asked to add new predicates on the attribute for which the dispersion of the initial set of answers is maximal. In this example, this approach may have suggested that the user should add a condition on the attributes *mileage* or *brand*, on which the recent estate cars are probably the most dispersed. We claim that it is more relevant to reduce the initial set of answers with additional conditions which are in the semantic scope of the initial query. Here for instance, it would be more judicious to focus on cars with a high level of security and comfort as well as a low mileage, which are features usually related to recent estate cars. This issue has been illustrated in Sect. 6.3.

The problem of plethoric answers to fuzzy queries has been addressed in [6] where a query strengthening mechanism is proposed. Let us consider a fuzzy set $F = (A, B, a, b)$ representing a fuzzy query Q. The authors of [6] define a fuzzy tolerance relation E which can be parameterized by a tolerance indicator Z, where Z is a fuzzy interval centered in 0 that can be represented in terms of a trapezoidal membership function by the quadruplet $Z = (-z, z, \delta, \delta)$. From a fuzzy set $F = (A, B, a, b)$ and a tolerance relation $E(Z)$, the erosion operator builds a set F_Z such that $F_Z \subseteq F$ and $F_Z = F \ominus Z = (A + z, B - z, a - \delta, b - \delta)$. However, such an erosion-based approach can lead to a deep modification of the meaning of the user query, if the erosion process is not correctly controlled.

8 Conclusion

This chapter is a synthesis of several works that we have carried out in the context of cooperative query answering. The main originality of our approach is that it addresses symmetrical problems with a unified framework based the notion of a *database fuzzy summary*. The type of summary we consider is based on fuzzy cardinalities and offers a concise formalism to represent the data distributions over a predefined vocabulary composed of fuzzy partitions. We have empirically shown on a concrete applicative context that this method is efficient and that it provides meaningful information that may help the user retrieve the items he/she is looking for. An important point is that the summarization process has a linear data complexity. On the other hand, this fuzzy-cardinality-based cooperative approach is realistic only when the number of predicates that compose the predefined fuzzy vocabulary is reasonably small. An interesting perspective would be to study the benefits of an incremental computation of the summaries bootstrapped with correlation between attributes or predicates that can be identified in a workload of previously submitted queries.

Concerning the failing query problem, we have proposed an approach that provides informative explanations about the reasons of the failure. A perspective is to define a strategy that automatically repairs the failing queries, the goal being to suggest a relaxed query that returns a non-empty set of answers and, if possible, whose cardinality is as close as possible to the quantitative parameter k. As mentioned above, an interesting solution could be to consider reformulations involving fuzzy quantified statements.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Bocca, J.B., Jarke, M., Zaniolo, C. (eds.) VLDB, pp. 487–499. Morgan Kaufmann, San Francisco (1994)
2. Bezdek, J.: Pattern Recognition with Fuzzy Objective Function Algorithm. Plenum Press, New York (1981)
3. Bodenhofer, U., Küng, J.: Fuzzy ordering in flexible query answering systems. Soft Comput. **8**, 512–522 (2003)
4. Bosc, P., Buckles, B., Petry, F., Pivert, O.: Fuzzy databases. In: Bezdek, J., Dubois, D., Prade, H. (eds.): Fuzzy Sets in Approximate Reasoning and Information Systems, pp. 403–468. The Handbook of Fuzzy Sets Series. Kluwer Academic Publishers, Dordrecht (1999)
5. Bosc, P., Dubois, D., Pivert, O., Prade, H., de Calmès, M.: Fuzzy summarization of data using fuzzy cardinalities. In: Proceedings of the 9th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'02), pp. 1553–1559, Annecy, France (2002)
6. Bosc, P., Hadjali, A., Pivert, O.: Empty versus overabundant answers to flexible relational queries. Fuzzy Sets Syst. **159**(12), 1450–1467 (2008)
7. Bosc, P., Hadjali, A., Pivert, O.: Incremental controlled relaxation of failing flexible queries. J. Intell. Inform. Syst. **33**(3), 261–283 (2009)
8. Bosc, P., Pivert, O.: SQLf: a relational database language for fuzzy querying. IEEE Trans. Fuzzy Syst. **3**(1), 1–17 (1995)

9. Bosc, P., Pivert, O., Dubois, D., Prade, H.: On fuzzy association rules based on fuzzy cardinalities. In: FUZZ-IEEE, pp. 461–464 (2001)
10. Bosc, P., Pivert, O., Hadjali, A., Smits, G.: Correlation-based query expansion. In: Actes des 26^{e} Journées Bases de Données Avancées (2010)
11. Chaudhuri, S., Das, G., Hristidis, V., Weikum, G.: Probabilistic ranking of database query results. In: Proceedings of VLDB'04, pp. 888–899 (2004)
12. Chomicki, J.: Querying with intrinsic preferences. In: Proceedings of EDBT'02, pp. 34–51 (2002)
13. Corella, F., Lewison, K.: A brief overview of cooperative answering. In: Technical report http://www.pomcor.com/whitepapers/cooperative_responses.pdf (2009)
14. Cuppens, F., Demolombe, R.: Cooperative answering: a methodology to provide intelligent access to databases. In: Proceedings of DEXA'88, pp. 333–353 (1988)
15. Dubois, D., Prade, H.: Fuzzy cardinalities and the modeling of imprecise quantification. Fuzzy Sets Syst. **16**, 199–230 (1985)
16. Dubois, D., Prade, H.: Fundamentals of fuzzy sets, volume 7 of The Handbooks of Fuzzy Sets. Kluwer Academic, The Netherlands (2000)
17. Gaasterland, T., Godfrey, P., Minker, J.: Relaxation as a platform for cooperative answering. J. Intell. Inform. Syst. **1**(3–4), 296–321 (1992)
18. Godfrey, P.: Minimization in cooperative response to failing database queries. Int. J. Cooperative Inform. Syst. **6**(2), 95–149 (1997)
19. Jannach, D.: Techniques for fast query relaxation in content-based recommender systems. In: Proceedings of KI'06, pp. 49–63 (2006)
20. Kaplan, S.-J.: Cooperative responses from a portable natural language query system. Artif. Intell. **19**, 165–187 (1982)
21. Kiessling, W.: Foundations of preferences in database systems. In: Proceedings of VLDB'02 (2002)
22. Zadeh, L.A.: Fuzzy sets. Inform. Control **8**(3), 338–353 (1965)
23. McSherry, D.: Incremental relaxation of unsuccessful queries. In: Proceedings of ECCBR'04, pp. 331–345 (2004)
24. McSherry, D.: Retrieval failure and recovery in recommender systems. Artif. Intell. Rev. **24**(3–4), 319–338 (2005)
25. Motro, A.: Cooperative database system. In: Proceedings of FQAS'94, pp. 1–16 (1994)
26. Ozawa, J., Yamada, K.: Cooperative answering with macro expression of a database. In: Proceedings of IPMU'94, pp. 17–22 (1994)
27. Ozawa, J., Yamada, K.: Discovery of global knowledge in database for cooperative answering. In: Proceedings of Fuzz-IEEE'95, pp. 849–852 (1995)
28. Pilarski, D.: Linguistic summarization of databases with quantirius: a reduction algorithm for generated summaries. Int. J. Uncertainty Fuzziness Knowl. Based Syst. **18**(3), 305–331 (2010)
29. Pivert, O., Bosc, P.: Fuzzy Preference Queries to Relational Databases. Imperial College Press, London (2012)
30. Pivert, O., Smits, G., Hadjali, A., Jaudoin, H.: Efficient detection of minimal failing subqueries in a fuzzy querying context. In: Eder, J., Bieliková, M., Tjoa, A.M. (eds.) ADBIS. Lecture Notes in Computer Science, vol. 6909, pp. 243–256. Springer (2011)
31. Ras, R.-W., Dardzinska, A.: Intelligent query answering. In: Wang, J. (ed.) Encyclopedia of Data Warehousing and Mining, 2nd edn, vol. II, pp. 1073–1078. Idea Group, Inc., Hershey (2008)
32. Rasmussen, D., Yager, R.R.: Summary SQL: a fuzzy tool for data mining. Intell. Data Anal. **1**(1–4), 49–58 (1997)
33. Ruspini, E.: A new approach to clustering. Inform. Control **15**(1), 22–32 (1969)
34. Saint-Paul, R., Raschia, G., Mouaddib, N.: General purpose database summarization. In: Proceedings of VLDB'05, pp. 733–744 (2005)
35. Su, W., Wang, J., Huang, Q., Lochovsky, F.: Query result ranking over e-commerce web databases. In: Proceedings of CIKM'06 (2006)

36. Ughetto, L., Voglozin, W.A., Mouaddib, N.: Database querying with personalized vocabulary using data summaries. Fuzzy Sets Syst. **159**(15), 2030–2046 (2008)
37. Zadeh, L.: A computational approach to fuzzy quantifiers in natural languages. Comput. Math. Appl. **9**, 149–183 (1983)