Supporting Failing Database Queries in a Flexible Context: A Data-Driven Approach

Lila Oudjoudi[1] and Allel Hadjali[2]

[1] ESI, BP 68M, 16270, Oued Smar, Algérie
l.oudjoudi@gmail.com

[2] IRISA/ENSSAT, University of Rennes 1
Technopole Anticipa 2205 Lannion Cedex France
hadjali@enssat.fr

Abstract. We investigate the problem of handling of failing queries involving fuzzy predicates. We propose an approach that leverages data distribution of the target database. It consists in two steps: i) Query translation that aims at translating the failing fuzzy query into a crisp query by means of a particular semantic distance between sets; ii) Query relaxation which consists in expanding the translated query criteria with similar values. To rank-order the approximate query results, a method is proposed

Keywords: Flexible queries, empty answers, semantic distance, similarity measures, relaxation.

1 Introduction

The practical need for endowing intelligent information systems with the ability to exhibit cooperative behavior has been recognized since the early' 90s. The most well-known problem approached in this field is the *failing query problem*: users' queries return an *empty set of answers*. Several approaches have been proposed to deal with this issue, see [9] for an overview. Most of them rely on the relaxation paradigm that aims at expanding the scope of a query searching for answers that are in the neighborhood of the original user's query.

On the other hand, relying on *flexible (or fuzzy) queries* (i.e., queries that could contain fuzzy constraints) has the main advantage of diminishing the risk of empty answers. Indeed, fuzzy queries express preferences and retrieve elements that are more or less satisfactory rather than necessarily ideal. However, it still may happen that the database does not have any element that satisfies, even partially, the fuzzy criteria formulated by the user. Only few works have been done for dealing with this problem in the fuzzy database querying [2][4][10][18]. They mainly aim at relaxing the fuzzy requirements involved in the failing query. *Query relaxation* can be achieved by applying an appropriate transformation to gradual predicates of a failing query. Such a transformation aims at modifying a given predicate into an enlarged one by *widening its support*. Recently, other kind of approaches which are based on

leveraging a past query workload (log of past user queries) have been proposed in [3][13]. The principle consists in replacing the failing query by the most similar one among the queries of the workload. All the approaches can be viewed as query-driven methods, i.e., they primarily operate on the failing query.

In this paper, we propose an approach for dealing with failing flexible conjunctive queries that leverages data distribution of the target database. It constitutes another direction to address the problem in a flexible context; the idea is somewhat close to the principle of similarity search. Instead of relaxing the failing initial query, we first look for the values in the database that are maximally close to the fuzzy predicates specified in that query and then we explore the neighborhoods of such values. Informally speaking, the approach proceeds in two steps: i) Query translation that aims at translating the failing fuzzy query into a (crisp) point query by means of a particular semantic distance measure between sets; ii) Query relaxation which consists in expanding the translated query criteria with similar values. To rank-order the query results, a ranking method is proposed.

The paper is structured as follows. Some basic notions are introduced in section 2. In section 3, we review some related work. Section 4 provides an overview of the approach proposed. Section 5 discusses the relaxation of both categorical and numerical query conditions. Section 6 describes a query results ranking method by learning attribute importance weights. In section 6, we conclude and outline some future works.

2 Basic Notions

2.1 Flexible Queries

Flexible queries [6] are requests in which user's preferences can be expressed. Here, the fuzzy sets framework is used as a tool for supporting the expression of preferences. The user does not specify crisp conditions (Boolean predicates), but fuzzy ones (which correspond to fuzzy predicates such as *Young*, *Tall* or *Cheap*) whose satisfaction may be regarded as a matter of *degree*. As a consequence, the result of a query is no longer a flat set of elements but is a set of discriminated elements according to their global satisfaction of the fuzzy constraints appearing in the query.

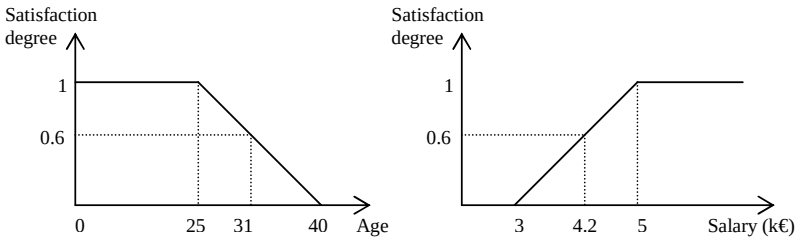

Fig. 1. Fuzzy predicates *Young = (0, 25, 0, 15)* and *Well-paid = (5, +∞, 2, 0)*.

An elementary fuzzy predicate P can be modeled as a function μ_P from a domain U to the unit interval. The degree $\mu_P(u)$ represents the extent to which element u satisfies the vague predicate P (or equivalently the extent to which u belongs to the fuzzy set of objects which match the fuzzy concept P). Here, we use trapezoidal membership functions (t.m.f.) that can be encoded by a quadruplet (A, B, a, b) where [A, B] (resp. [A-a, B+b]) represents the core (resp. the support) of P. A typical example of a fuzzy query is: "retrieve the employees which are *Young* and *Well-paid*", see Figure 1.

2.2 Semantic Distance

We introduce here a particular semantic distance between fuzzy (or crisp) sets. It relies on the *Hausdorff distance measure* whose principle is reviewed hereafter.

***2.2.1 Crisp Sets*:** Consider two subsets A and B of a space U (equipped with a metric). The most popular scalar extension of distance between A and B is the *Hausdorff distance* defined as [7][11]:

$$d_H(A, B) = max\{H(A, B), H(B, A)\}, \quad (1)$$

where $H(A, B)$ stands for the *directed Hausdorff distance* from A to B. We have $H(A, B) = Sup_{u\in A}\ d(u, B)$ and $d(u, B) = Inf_{v\in B}\ d(u, v)$. The expression $d(u, v)$ stands for a standard distance (such as Euclidean distance). Formula (1) can be written in the following condensed form:

$$d_H(A, B) = max\{sup_{u\in A}\ inf_{v\in B}\ d(u, v), sup_{v\in B}\ inf_{u\in A}\ d(u, v)\}. \quad (2)$$

The idea that governs this distance is the following: for each element in A look for the closest element in B, then check for the element in A for which the distance to the closest element in B is maximal. The same is done exchanging B and A and the *longest* distance of the two component is kept. Intuitively, if the Hausdorff distance is δ, then every point of A must be within a distance δ of some point of B and vice versa.

Example 1. Let $A = [a_1, a_2]$ and $B = [b_1, b_2]$ be two regular intervals and let $d(u, v) = |u - v|$. Then, it easy to check that $d_H(A, B) = max(|a_1 - b_1|, |a_2 - b_2|)$. ♦

***2.2.2 Fuzzy Sets*:** The Hausdorff distance between fuzzy sets can be either fuzzy or scalar. Hereafter, we only focus on the scalar version. For the fuzzy evaluation, more details are available in [11]. Here, we use the definition proposed in [7]. This definition is more general and is valid in the case of two fuzzy sets with unequal maximum memberships. In the following, we consider only fuzzy sets with the same supremum.

Let F and G be two discrete fuzzy sets. Let $T = \{t_1, t_2, \ldots, t_m\}$ the set of all the distinct membership values of F and G. The Hausdorff distance between F and G is defined by the following expression:

$$d_H^2(F,G) = \frac{\sum_{i=1}^{m} t_i d_H(F_{t_i}, G_{t_i})}{\sum_{i=1}^{m} t_i}\ , \quad (3)$$

where F_{t_i} (resp. G_{t_i}) stands for the t_i-level cut[1] of F (resp. G). $d_H^2(F,G)$ can be seen as a membership-weighted average of the crisp Hausdorff distances between the level sets of the two fuzzy sets.

Example 2. Let *U = {1, 2, 3, 4, 5, 6, 7}* be a universe of discourse. Let also F and G be two discrete fuzzy sets on U defined as follows: *F = {0.7/1, 0.2/2, 0.6/4, 0.5/5, 1/6}* and *G = {0.2/1, 0.6/4, 0.8/5, 1/7}*. One can see that *T = {0.2, 0.5, 0.6, 0.7, 0.8, 1}*.

Table 1. The Hausdorff distance between the α-cuts of F and G

α_i	F_{α_i}	G_{α_i}	$H(F_{\alpha_i}, G_{\alpha_i})$	$H(G_{\alpha_i}, F_{\alpha_i})$	$d_H(F_{\alpha_i}, G_{\alpha_i})$
0.2	{1, 2, 4, 5, 6}	{1, 4, 5, 7}	1	1	1
0.5	{1, 4, 5, 6}	{4, 5, 7}	3	1	3
0.6	{1, 4, 6}	{4, 5, 7}	3	1	3
0.7	{1, 6}	{5, 7}	4	1	4
0.8	{6}	{5, 7}	1	1	1
1	{6}	{7}	1	1	1

By formula (3), and using Table 1, we get

$d_H^2(F,G) = (0.2 \cdot 1 + 0.5 \cdot 3 + 0.6 \cdot 3 + 0.7 \cdot 4 + 0.8 \cdot 1 + 1 \cdot 1) / 3.8 \cong 2.13$ ♦

In case of continuous fuzzy sets, formula (3) is modified in the following form [7]:

$$d_H^2(F,G) = \int_0^1 t d_H(F_t,G_t)dt \Big/ \int_0^1 t dt = 2\int_0^1 t d_H(F_t,G_t)dt \qquad (4)$$

Example 3. Let now U represent the numeric universe of discourse of the variable "*age*" of a person. Let also *F* = "*about thirty*" and *G* = "*between_26_and_28*" two fuzzy sets on U defined by the following two *t.m.f.*: *F = (30, 30, 3, 3)* and *G = (26, 28, 1, 1)*. One can observe that $F_\alpha = [3\alpha + 27, 33 - 3\alpha]$ and $G_\alpha = [\alpha + 25, 29 - \alpha]$. Then, Applying formula (4), we get

$d_H^2(F,G) = 2\int_0^1 t \max(|(t+25)-(3t+27)|, |(29-t)-(33-3t)|)dt = 7/2$ ♦

It has been pointed out in [7] that expression (3) (resp. (4)) is a metric and reduces to the classical Hausdorff distance when sets are crisp.

3 Related Work

Several Works have been proposed to deal with the empty answers problem. Such works can be found in both domains of databases and information retrieval, including web search. Due to space limitation, we only provide here a review on some approaches proposed in the database fuzzy querying context. See [9][4][16][14] for an overview of the approaches suggested in the crisp queries context.

In the fuzzy querying setting, approaches can be classified into two main categories and are mainly query-driven. The first one is based on the relaxation

[1] An α-levet cut of the fuzzy set F is defined as $\{u \in U, \mu_F(u) \geq \alpha\}$.

paradigm. Query relaxation aims at expanding the scope of a query searching and consists in modifying some query conditions by enlarging them or just eliminating some of them. Andreasen and Pivert [2] have proposed an approach where the basic modification used relies on a particular *expansive linguistic modifier*. This approach is merely a technical operation, lacking of any semantics. Moreover, it provides no intrinsic *semantic limits* for controlling the relaxation process and fails to deal with classical crisp queries. In [5][4] a relaxation method is proposed that makes use of a parameterized proximity relation. Given a fuzzy predicate P, the idea is to compute the set of predicates that are close to P in the sense of the proximity relation defined on the domain of P. Even, if this method is endowed with a clear semantics, it can lead to a combinatory explosion induced by the relaxation of the predicates from a conjunctive query. To know whether these relaxed queries provide a non-empty answer, one has to evaluate them. In [18], the authors consider flexible queries addressed to data summaries and propose a method based on a specified distance to repair failing queries. If no summary fits a query *Q*, alternative queries are generated by modifying one or several fuzzy labels involved in *Q*. This requires a *pre-established order* over the considered attributes domains since a label is replaced by the closest one. The resulting queries are ordered according to their closeness to the original one (measured by the specified distance). See also the work done in [10].

The second category is based on leveraging a past query workload (i.e., a collection of queries that have been executed on the database system in the past and have produced non-empty answers). The principle consists in replacing the failing query by the most similar (semantically speaking) one among the queries of the workload. To compute the proximity between queries, a measure of substitution is suggested in [3] which assumes the availability of a resemblance relation over every attribute domain involved in the target database. An alternative proximity query measure is studied in [13]. It relies on a particular distance between sets, called the Hausdorff distance. Only attributes with domains endowed with a metric have been considered in this work.

Our work is inspired from [1] and [17] for computing the importance weights for each specified attribute and for deriving the similarity coefficients between two (categorical or numerical) values. In [1], an automatic ranking method based on Information Retrieval (IR) techniques for the empty answers problem is proposed. The importance scores of tuples are extracted using a workload and a data analysis. In [17], a system called AIMQ is proposed to address the problem of answering imprecise queries over Web databases. It learns attribute importance and values similarity measures from the database. It can only determine the attribute importance sequence (without the specific weights). This result is invariant for the different user queries. See also [12] for the incremental version of AIMQ, called IQPI. Let us mention the work done in [15] that uses, in a similar way as above, data and query workload statistics for relaxing crisp queries in order to provide approximate answers to the user.

Our approach differs from that in [2][3][4][5][13] in leveraging only data distribution for relaxing failing queries, and from [1][17][15] in focusing on fuzzy queries.

4 Overview of the Approach

Let us first state the problem of interest. Assume that $\mathcal{D}$ is a (Web) regular database with categorical and numerical attributes $A = \{A_1, \ldots, A_m\}$ and $D(A_i)$ represents the domain of values of attribute A_i in the database $\mathcal{D}$. Let also $Q = P_1 \wedge \ldots \wedge P_k$ $(k < m)$ be a conjunctive flexible query where the symbol '$\wedge$' stands for the connector *'and'* and is interpreted by the '*min'* operator. Let Σ_Q be the set of answers to Q over $\mathcal{D}$. The set Σ_Q contains the items of the database that satisfy *somewhat* the fuzzy requirements involved in Q, i.e., each item has a strict positive satisfaction degree.

Definition. We say that Q is a failing query if $\Sigma_Q = \varnothing$.

This means that no data in the database somewhat satisfies all of the fuzzy conditions involved in Q. In the literature, this problem is known as the *Empty Answers Problem*.

Let us assume that Q is a failing user query. To deal with this problem, one way is to provide approximate answers to the user. To this end, we propose a data-driven approach that leverages the data distribution of the target database. It consists in a two-step procedure (see Figure 2):

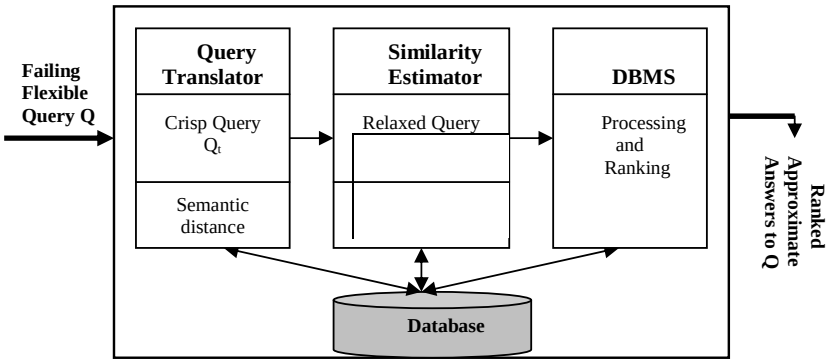

Fig.2. Architecture of the approach

Step 1: Query Translation. It aims at translating Q into a (crisp) point query of the form:

$$Q_t = v_1 \wedge \ldots \wedge v_k,$$

where $v_i \in D(A_i)$. For each P_i, $i = 1, \ldots, k$, pertaining to attribute A_i, we look for the point $v_i \in D(A_i)$ that is maximally close, semantically speaking, to the fuzzy set modelling P_i. To do so, we assess the semantic distance between the set E_i and P_i where the t.m.f. of E_i is given by $(v_i, v_i, 0, 0)$ (resp. $\{1/ v_i\}$) if P_i is a numerical (resp. categorical) attribute. This can be achieved by computing the Hausdorff distance between E_i and P_i (discussed in Section 2.2).

Example 4. To illustrate this step, consider a relation *EMP(Name, Age, Salary)* describing employees of a company, whose extension is given in Table 2. Consider also the query Q = *"find employees who are young and earn around 38 k$"*. Q can

simply write $Q = (Age = Young) \wedge (Salary = Around\_38)$ where *Young* and *Around_38* are fuzzy sets represented respectively by the t.m.f. *(0, 25, 0, 10)* and *(38, 38, 2, 2)*. One can observe that Q returns an empty answer when evaluating over relation EMP of Table 2.

Now according to step 1 and using the Hausdorff distance between each *Age.value* (resp. *Salary.value*) and *Young* (resp. *Around_38*), *Q* is rewritten in $Q_t = (Age = 38) \wedge (Salary = 35)$. ♦

Table 2. Extension of Relation EMP

Name	**Age**	**Salary** (k$)	$\mu_{Young}(u)$	$\mu_{Around\_38}(v)$
Dupont	48	45	0	0
Martin	46	42	0	0
Durant	42	35	0	0
Dubois	38	28	0	0
Lorant	40	30	0	0

Now according to step 1 and using the Hausdorff distance between each *Age.value* (resp. *Salary.value*) and *Young* (resp. *Around_38*), *Q* is rewritten in $Q_t = (Age = 38) \wedge (Salary = 35)$. ♦

Remark. If there are several values $\{v_i^1, \ldots, v_i^h\}$ from $D(A_i)$ that are maximally close to P_i, we translate the fuzzy query condition $(A_i \text{ is } P_i)$ into $(A_i \in \{v_i^1, \ldots, v_i^h\})$.

Step 2: Query Relaxation. We rewrite Q_t under the form

$$Q_t = C_1 \wedge \ldots \wedge C_k$$

where $C_i = (A_i = v_i)$ for $i = 1,\ldots, k$. Then, for each condition C_i in Q_t, we extract values of its corresponding attribute A_i having similarity factor above some user-defined sub-threshold λ_i. Then, we add those values into the range of C_i. Thus, we get a relaxed version of C_i, denoted by $\widetilde{C_i}$. A relaxed version, denoted by $\widetilde{Q_t}$, of the query Q_t is then built by joining all the relaxed conditions $\widetilde{C_i}$.

Full details about the above two steps will be provided in the next sections.

5 Query Relaxation

The idea of query relaxation advocated consists in expanding the translated query criteria with similar values. So, we need to measure the similarity between the different pairs of values.

5.1 Relaxation of Categorical Query Conditions

We present an approach for measuring the similarity index between two categorical attribute values. This approach is borrowed from [17]. The similarity between two

values binding a categorical attribute is measured as the percentage of common *Attribute-Value pairs* (AV-pairs) that are associated to them. Consider a used car selling Web database *CarDB(Make, Model, Price, Color, Year).* Each tuple in *CarDB* represents a used car for a sale. For instance, *Make = Ford* is AV-pair over the database CarDB.

- Each AV-pair is considered as a selection query and submitted to (a sample of) the database, separately.
- The result of running each query is a set of tuples which is called a *supertuple (ST).*
- The supertuple contains a bag of keywords for each attribute in the relation not bound by the AV-pair.

For example, Table 3 shows the supertuples of the AV-pair "*Make = Toyota*" and "*Make = Ford*" over the database *CarDB.*

Table 3. The supertuples obtained from running

(a) the query "Make = Toyota".

Model	Camry: 3, Corolla: 4
Price	10k-15k: 4 , 15k-20k: 3
Color	Blue: 1 , Black: 3 , White : 3
Year	2005: 2 , 2006: 3 , 2007 : 2

(b) the query "Make = Ford"

Model	Focus: 2, F150: 3
Price	10k-15k: 3 , 15k-20k: 2
Color	Blue: 2 , Red: 2 , White : 1
Year	2005: 1 , 2006: 4

The values within the supertuple of Table 3-(a) indicate that there are totally *7* records in the database having "*Make = Toyota*". ♦

The similarity between two attribute values (AV-pairs) is measured as the similarity shown by their supertuples. This latter is measured by using the *Jaccard* coefficient. Let ST_1 and ST_2 be two supertuples with *m* attributes and A_i is i^{th} attribute, we have

$$VSim(ST_1, ST_2) = \sum_{i=1}^{m} J(ST_1.A_i, ST_2.A_i),$$

where *J(.,.)* stands for the Jaccard Coefficient and is computed as *J(A, B)* = $/A \cap B/ / /A \cup B/$. Consider for instance the attribute "*Make*", if we want to measure the similarity between the value "*Toyota*" and the value "*Ford*". First, we compute the supertuple ST_1 (resp. ST_2) resulting from the query "*Make = Toyota*" (resp. "*Make = Ford*)" (See Table 3). Then, we compute $VSim(ST_1, ST_2)$ = (2·0)/(4·7) + (2·5)/(4·7) + (2·1)/(3·10) + (1·4)/(4·8) ≈ 0.54 [2]. So, *VSim(Toyota, Ford) =* $VSim(ST_1, ST_2)$ *= 0.54.*

[2] Here, we use *Jaccard Coefficient* with bag semantics to determine the similarity between two supertuples, see [17].

Now if needed, one can normalize the above similarity measure by using for instance the arithmetic mean, i.e., $VSim(ST_1, ST_2) = \frac{1}{m}\sum_{i=1}^{m} J(ST_1.A_i, ST_2.A_i)$.

5.2 Relaxation of Numerical Attribute Values

To estimate the similarity coefficient between a pair of different numerical values, we use an approach which is inspired from [8][1]. Let $\{a_1,.., a_n\}$ be the values of numerical attribute A occurring in the database. Then the similarity coefficient $VSim(a, v)$ between the two values a and v can be defined by the following equation

$$VSim(v, a) = 1 /(1 + ((a - v)/h)^2)$$

where h is the bandwidth parameter. A popular estimation for the bandwidth is $h = 1.06\sigma n^{-1/5}$ where σ is the standard deviation of $\{a_1, ….., a_n\}$, see [1] for more details. Let λ be a given similarity threshold, and v the numerical value specified by the query. Then, one can observe that the values that have similarity degree (above λ) with v are restricted by the following interval:

$$I(v,\lambda) = \left[v - h\sqrt{(1-\lambda)/\lambda}, v + h\sqrt{(1-\lambda)/\lambda}\right]$$

Input: $Q_t = \{C_1, …, C_k\}$ with $C_i = (A_i = v_i)$ for $i = 1,k$
Sub-thresholds $\{\lambda_1, …, \lambda_k\}$

1. $\tilde{Q}_t = \emptyset$; $i := 1$;
2. **while** $i \leq k$ **do**
3. **begin**
5. $\tilde{C}_i := C_i$;
6. **if** A_i is numerical attribute **then**
7. replace the range of $\tilde{C}_i$ with $I(v_i, \lambda_i)$;
8. **if** A_i is categorical attribute **then**
9. **For** a **in** $D(A_i)$ **do**
10. **If** $VSim(v_i, a) = VSim(ST(v_i), ST(a)) > \lambda_i$ **then**
11. add a into the range of
12. **endif**
13. **endif**
14. $\tilde{Q}_t := \tilde{Q}_t \cup \tilde{C}_i$
15. $i := i + 1$;
16. end

Output: the query relaxation $\tilde{Q}_t = \tilde{C}_1 \wedge \ldots \wedge \tilde{C}_k$

Algorithm 1. Query relaxation (where $ST(v)$ is the *supertuple* associated with the value v).

5.3 Query Rewriting

Let us assume that the sub-threshold λ_i *(i = 1, k)* for each specified attribute in *Q* is given by the user. The principle of the query relaxation algorithm (see Algorithm 1) is to replace each condition $C_i = (A_i = v_i)$ involved in Q_t by its relaxed variant, $\widetilde{C}_i$, as explained in Sections 5.1 and 5.2.

Remark. If the relaxed query, $\widetilde{Q}_t$, resulting from Algorithm 1 still returns an empty answer set, one can re-execute Algorithm 1 by assigning other appropriate values to the sub-threshold λ_i.

Note that for large-scale databases, the evaluation of the relaxed query may result in too many relevant answer items. So, it is extremely desirable to rank such query results according to their relevance. This is what we will discuss in the next section.

6 Results Ranking Strategy

One factor that can affect query results ranking is the attribute importance weights (since attribute importance of the same attribute is usually different for users). It would then be interesting to automatically learn such importance weights. Approaches for estimating attribute importance can be divided into two classes [17]: (i) *data driven* where the attribute importance is identified using the data distribution of the database; (ii) *query driven* where the importance of an attribute is determined by the frequency with which it appears in user queries. So, this last technique requires a database workload (log of past user queries) which constrains its use for new systems. In the following, we use the first technique to learn the importance of each attribute by leveraging the distribution of its value specified in the query in the database.

6.1 Attribute Weight Assignment

The idea is to associate a weight to each specified attribute according to the distribution of its value in the database. For instance, for a query with condition "*Year = 2008* and *Price < 10000*", the specified attribute *Year* may have less importance for user (there may be many used car have the date of shipment in 2008) than the attribute *Price* (relatively fewer used cars priced below $10000).

To this end, we use the well-known Inverse Document Frequency (*IDF*) factor that has been used extensively in IR. *IDF* suggests that commonly occurring words convey less information about user's needs than rarely occurring words, and thus should be weighted less. Recall that *IDF(w)* of a word *w* is a measure indicating how many documents in which *w* appears. We can then adapt this technique to our problem by considering each database tuple (and query) as a small document [1].

- Categorical Attribute

For every *v* in the domain of attribute A_i, we define $IDF_i(v)$ such that

$$IDF_i(v) = log(n/F_i(v)),$$

where n is the number of tuples in the database and $F_i(v)$ is the frequency of tuples t in the database where $t.A_i = v$. In the following, the importance of specified categorical attribute value is treated as the importance of its corresponding attribute.

- Numerical Attribute

For numerical data, the definition of traditional *IDF* as above is inappropriate. The frequency (and hence the *IDF*) of a numerical value should depend on nearby values. To measure the closeness of numerical attribute values, we make use of a robust definition proposed in [1]. Let $V_i = \{v_i^1, \ldots..., v_i^n\}$ be the set of values of attribute A_i that occur in the database. For any value v, $IDF_i(v)$ is defined in the following way (where h is the bandwidth parameter mentioned in section 5.2):

$$IDF_i(v) = log\left(\frac{n}{\sum_{j=1}^{n} e^{-\frac{1}{2}\left(\frac{v_i^j - v}{h}\right)}}\right)$$

The intuition of the above formula is: the denominator represents a numeric extension of the concept of frequency of v (the sum of "contributions" to v from every the other point v_i^j in the database). The further v is from v_i^j, the smaller its contribution. The importance of specified numerical attribute value is treated as the importance of its corresponding attribute

Denoting by $W_i(A_i)$ the weight associated with attribute A_i specified in the user query and by applying a normalization, we finally obtain:

$$W_i(A_i) = \frac{IDF_i(v_i)}{\sum_{j=1}^{k} IDF_j(v_j)}$$

6.2 Ranking

To rank-order the answer tuples t returned to the relaxed query, one can use the following ranking score:

$$d_{Q_t}(t) = \frac{1}{k}\sum_{i=1}^{k} W_i(A_i) \times Sim(Q_t.A_i, t.A_i),$$

where $W_i(A_i)$ is the importance weight of attribute A_i specified in the original query Q, and $Sim(v, a)$ stands for the similarity measures between categorical or numerical attribute values as explained in section 5.

The idea is that the larger the similarity score $d_Q(t)$ is, the higher the ranking score is for the result tuple t. Thus, we can provide the end-user with the top-N answers according to the above ranking.

7 Discussion and Conclusion

In this paper, an alternate approach for dealing with failing queries in a flexible context is proposed. Starting from the user query, we translate the original query into a crisp query and then we rewrite this resulting query by relaxing the query criteria ranges. The two key concepts of the approach are the distance semantic introduced and the similarity measures discussed (between two categorical or numerical attribute values). Ranking the relevant answers is also investigated by learning the importance of each attribute specified in the query. Since the approach mainly operates on the actual data of the queried database, it could constitute a promising alternative to approximately answer a failing query.

As can be seen, the approach requires some pre-computations and some additional access to the entire database before getting final approximate answers to the failing query. Pre-computations mainly consists in: i) calculating the semantic distance between attribute values in the database and the fuzzy predicate present in the query for every attribute specified in the query; ii) measuring the similarity between each value of attribute involved in the translated query and values binding this attribute in the database. Let us take a look at the second type of calculus in case of categorical attribute. Instead of performing this calculus for each failing query, one can measure the similarity between every pair of values binding this attribute once and for all (provided that database will not change), and the similarity results will be stored in a Table. To mitigate also the problem of scanning the entire database, one can select a sample dataset (sufficiently representative) for estimating the similarity measures.

We acknowledge that experiments on real databases are needed to demonstrate the efficiency and effectiveness of the approach. To this end, one can observe that at the end of the query answering process, it is a precise (point or range) query ($\tilde{Q}_t$) that is evaluated over the database. So, one can implement the approach over (traditional) existing database systems. To such existing systems, two modules have to be added (see Figure 2) (i) a query translator that converts the flexible query into a crisp query; (ii) a similarity estimator that computes the similarity measures between attribute values. Experiments to perform allow also for providing an idea about the extra cost resulting from the use of the approach. Besides, only attributes with domains endowed with a metric can be addressed by the approach. It would be interesting to extend it to attributes with non metricized domains (as *color* attribute).

References

1. S. Agrawal, S. Chaudhuri, G. Das, V. Hristidis, A. Gionis, Automated ranking of database query results. CIDR 2003.
2. T. Andreasen, O. Pivert, On the weakening of fuzzy relational queries, in *8th Int. Symp. On Meth. for Intell. Syst.*, Charlotte, USA, 1994, pp. 144-151.
3. P. Bosc, C. Brando, A. Hadjali, H. Jaudoin, O. Pivert, "Semantic proximity between queries and the empty answer problem", In IFSA World Congress, Lisbon, Portugal, 2009.
4. P. Bosc, A. Hadjali, O. Pivert, Incremental Controlled Relaxation of Failing Flexible Queries, *Journal of Intelligent Information Systems*, Vol. 3(3), 2009, pp. 261-283.

5. P. Bosc, A. Hadjali, O. Pivert, Empty versus Overabundant Answers to Flexible Queries, *Fuzzy sets and Systems Journal*, Vol. 159(16), 2008, pp. 1450-1467.
6. P. Bosc and O. Pivert.: SQLf : a relational database language for fuzzy querying. IEEE Transactions on Fuzzy Systems, vol. 3(1),pp. 1–17, 1995.
7. B.B. Chaudhuri and A. Rosenfeld, "A modified Hausdorff distance between fuzzy sets", *Information Sciences*, Vol. 118, pp. 159-171, 1999.
8. V. Cross and T. Sudkamp, "Similarity and Compatibility in Fuzzy Set Theory: Assessment and Applications", Studies in Fuzziness and Soft Computing, No 93, Physica-Verlag, 2002.
9. F. Corella, K.P. Lewison, A brief Overview of Cooperative Answering. Technical Report, http://www.pomcor.com/whitepapers/cooperative_responses.pdf, August 2009.
10. M. De Calmès, D. Dubois, E. Hullermeier, H. Prade, and F. Sedes, F, Flexibility and fuzzy case-based evaluation in querying: An illustration in an experimental setting. *Int. Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 11(1), 43-66, 2003.
11. D. Dubois and H. Prade, "On distances between fuzzy points and their use for plausible reasoning", In Proc. Int. Conf. on Systems, Man and Cybernetics, 1983, pp. 300-303.
12. S.M. Fakhr Ahmad, M.H. Sadreddini, M. Zolghadri Jahromi, IQPI: An incremental system for answering imprecise queries using approximate dependencies and concept similarities. Journal of Computer Sciences, 34(2), 2007.
13. A. Hadjali, Providing Approximate Answers to Flexible Failing Queries. Technical Report, September 2009.
14. G. Luo, Efficient detection of empty-result queries. VLDB '06, pp. 1015-1025, 2006.
15. X. Meng, ZM. Ma, L. Yan, Providing flexible queries over Web databases. KES'08, pp. 601-606, 2008.
16. C. Mishra, N. Koudas, Interactive query refinement. EDBT'09, pp. 862-873, 2009.
17. U. Nambiar, S. Kambhampati, Answering Imprecise Queries over Autonomous Web Databases. ICDE'06, pp. 45-54, 2006.
18. W.A. Voglozin, G. Rashia, L. Ughetto and N. Mouaddib, Querying the SaintEtiq summaries: Dealing with null answers. Proc. *IEEE Inter. Conf. on Fuzzy Systems*, pp. 585-590, USA, 2005.