Semantic Proximity Between Queries and the Empty Answer Problem

P Bosc, C Brando, H Hadjali, Hélène Jaudoin, O Pivert

Semantic Proximity Between Queries and the Empty Answer Problem

P. Bosc, C. Brando, A. Hadjali, H. Jaudoin, O. Pivert

IRISA – ENSSAT
Université de Rennes 1
BP 80518
22305 Lannion Cedex
France
Email: {bosc | cbrando | hadjali | jaudoin | pivert}@enssat.fr

Abstract—This paper proposes an approach aimed at obviating empty answers for a family of conjunctive queries involving Boolean or fuzzy value constraints. Contrary to the approaches based on a relaxation of the predicates involved in the query, the principle suggested here consists in replacing the query by another one similar which has been processed previously and whose answer is known to be non-empty. This technique thus avoids the combinatory induced by classical relaxation-based approaches.

Keywords—databases, fuzzy queries, empty answers, proximity.

1 Introduction

Since the late 80's, there is an increasing interest in designing intelligent information systems endowed with some cooperative behavior [12]. The most well-known issue approached in this field is the *empty answer problem*, that is, the problem of providing the user with some alternative data when there is no data fitting his/her query. Several approaches have been proposed to deal with this issue. The *relaxation paradigm* [13] is one of the basic cooperative techniques used in most of such approaches. Query relaxation aims at expanding the scope of a query searching for answers which are in the neighborhood of the user's query and consists in replacing some query conditions by more general conditions or in just eliminating some of them.

Let us note that manually relaxing failing queries is a tedious and time-consuming task because, in the worst case, one must consider an exponential number of possible relaxations [21]. Hence, several automated approaches to query relaxation have been proposed (see, e.g., [8, 11, 15, 17, 20, 21, 23). The main objective of those approaches is to modify a failing user query into a relaxed query whose answer is non-empty, or at least to identify the cause of the failure. Some of those works rely on the key concept of *false presuppositions* (a presupposition of a statement is any statement entailed by the original, for instance, the statement "the king of France is bald" has as presupposition "there is a king of France" which is a *false* presupposition). Motro [20] has addressed the issue of empty answers, i.e. when a query fails to produce any answers, by proposing a relaxation method which focuses on finding the false presuppositions of a failing query. A related approach has been proposed by Godfrey [15], who considers any subquery as a presupposition of the query itself. The focus of this work is the search for *Minimal Failing Subqueries* (MFSs) of a failing query.

In this paper, we propose an approach for dealing with failing conjunctive queries (Boolean or fuzzy), according to another approach than relaxation as described above. The idea that we advocate is not to modify/suppress some predicates from a failing query Q using this sole query, but rather to find a "good" global substitute to Q among queries previously submitted to the system whose answer is known to be non-empty. We thus consider a context where the system stores the non-failing queries in a repository D^+. One also assumes available a resemblance measure over every attribute domain involved in the database considered. The approach raises the question of defining the notion of *semantic proximity* between queries. With respect to related works, which will be briefly presented farther, the main originality of the approach introduced here is to take into account queries involving Boolean or fuzzy *value constraints* in an explicit form.

An important gain brought by this method, relatively to a classical relaxation-based approach, lies in the fact that it avoids the combinatory induced by the relaxation of the predicates from a conjunctive query. Indeed, there exists in general a high number of relaxed queries and one cannot know whether these queries provide a non-empty answer before processing them. With the approach proposed here, one has the guarantee to obtain a non-empty answer in one step because only one query needs to be processed.

In this paper, we limit the scope to conjunctive selection queries involving value constraints which can be represented either by Boolean sets, intervals or fuzzy sets.

The remainder of the paper is organized as follows. Section 2 presents a query substitution approach in the case where the predicates expressing value constraints are Boolean. Section 3 generalizes this approach to the case of conjunctive fuzzy queries. In Section 4, we describe the principle of a mechanism aimed at providing the user with some (at least partial) explanations about the causes of the original empty answer. The technique proposed uses both the repository D^+ and a second one denoted by D^- which contains the *failing queries* previously submitted to the system. Section 5 is devoted to a comparison of the approach with some related works. Finally, the conclusion recalls the main contributions of the paper and outlines some perspectives for future work.

2 Boolean queries

2.1 Single-predicate queries

Let Q = "A in E" denote the user query where A is an attribute and E a set or an interval, and Q' a query from the repository D^+. We denote by r the database relation concerned by Q and by res the proximity relation defined over the domain of A.

2.1.1 Case where E is a set

It is assumed that the user query Q returns an empty answer. One thus has to search D^+ so as to retrieve the queries involving a predicate of the form (A in E').

Remark 1. Any set E' present in the queries from D^+ is such that $E' \not\subseteq E$. Otherwise the associated query Q' would have returned an empty answer and Q' would not be in D^+.

The emptiness of the answer to Q means that none of the elements from E is present as an A-value in relation r. In order to obtain a substitute to Q which returns a non-empty answer, it is thus necessary to find a query Q' bearing on r and involving a predicate (A in E') such that E' contains at least one value absent from E. However, so as not to drift too far away from the initial user need, it is desirable that the elements from E' absent from E be sufficiently close to at least one element from E. Consequently, one looks for the set E' which is as close as possible to E, so as to replace E by (E' – E) in Q. In order to find this "best" E', a measure is needed. It is not strictly speaking a proximity measure, since the symmetry property is not desired here. Indeed one wants to know whether E' is a good substitute to E, but not the reciprocal. Several possible substitutivity measures are discussed hereafter.

1st idea: one assesses the extent to which every element from (E' – E) resembles at least one element from E:

$$sbs1(E, E') = \inf_{x' \in (E' - E)} \sup_{x \in E} res(x, x')$$

The problem with this measure is that the worst element "masks" the others, as illustrated in the next example. In the following, we assume available the following subset of a resemblance relation on animals:
res(rooster, hen) = 0.9,
res(rooster, duck) = 0.6,
res(rooster, turkey) = 0.7,
res(hen, turkey) = 0.7,
res(cow, hen) = res(cow, duck) = res(cow, turkey) = 0.

Example 1. E = {hen, duck, turkey}, E_1 = {hen, turkey, cow}, E2 = {cow, rooster}. We get $sbs1(E, E_1) = sbs1(E, E_2) = 0$ but since there are neither hens nor turkeys in the database – otherwise Q would not be failing –, it seems reasonable to claim that E_2 should be a better substitute than E_1. However, in the computation of sbs1(E, E2), the element cow "masks" rooster.♦

2nd idea: one assesses the extent to which there is an element from (E' – E) which resembles at least an element from E:

$$sbs2(E, E') = \sup_{x' \in (E' - E)} \sup_{x \in E} res(x, x')$$

Here, the difficulty is that the "winning set" may include elements which are very distant from those desired by the user.

Example 2. E = {hen}, E_1 = {hen, cow, turkey}, E_2 = {duck}. Here, E_2 should win, since it includes only elements close to the desired ones, contrary to E_1, which includes cow. However, it is E_1 which wins since $sbs2(E, E_1) = 0.7$ while $sbs2(E, E_2) = 0.6$.♦

3rd idea: one mixes the quantitative and the qualitative aspects by measuring the average resemblance degree between an element from (E' – E) and an element from E. For each element x' from (E' – E), the corresponding measure looks for the maximal proximity between x' and an element x from E, computes the sum of these maximal proximities, and divides this sum by the number of elements present in (E' – E).

$$sbs3(E, E') = [\Sigma_{x' \in (E' - E)} \sup_{x \in E} res(x, x')] / |E' - E|.$$

Example 3. E = {hen, duck, turkey}, E_1 = {hen, turkey, cow}, E_2 = {cow, rooster}. We get:

$sbs3(E, E_1) = 0$ and $sbs3(E, E_2) = 0.45$.♦
Since measure sbs3 appears the most satisfactory, it will be used in the following.

2.1.2. *Case where E is an interval*

It is quite straightforward to extend measure sbs3 defined previously so as to make it work with intervals instead of sets: one just has to replace the sum by an integral. The calculus is rather simple since it boils down to computing areas of rectangles or trapezoids.
Let us first consider the simple case where resemblance is defined in a Boolean manner:

res(x, y) = 1 if $|x - y| \leq \delta$, 0 otherwise.

Let us consider two intervals: I = [m, M], that from the user query, and I' = [m', M'], that from the candidate substitute. Let us assume that $m \leq m'$, The dual case can be obtained straightforwardly from this one. One gets:

sbs3(I, I') = 0 if $M + \delta \leq m'$
= 1 if $M' \leq M + \delta$
= $(M + \delta - m')/(M' - m')$ otherwise.

A slightly more complex case is that where resemblance is defined by means of a fuzzy tolerance indicator Z with a trapezoidal membership function of support $[-\alpha, \alpha]$ and of core $[-\beta, \beta]$. In other words:

$res(x, y) = \mu_Z(|x - y|)$.

The principle is the same as for Boolean resemblance, except that one has to compute areas of trapezoids instead of rectangles.
The case where the predicate from Q involves an interval I and that from Q' involves a set E can be managed by rewriting the definition of sbs3 the following way:

$$sbs3(I, E) = \Sigma_{x' \in E \text{ et } x' \notin I)} \sup_{x \in I} res(x, x') / |\{x \in E \mid x \notin I\}|.$$

On the other hand, the dual case (set in Q and interval in Q') is more tricky and cannot be captured by the formula

defining sbs3 when the attribute domain is continuous. Consequently, we introduce the constraint that a set can only be replaced by another set.

2.2 Conjunctive queries

Let Q be the user query and Q' a query from the repository D^+.

Remark 2. Even if query Q' involves a predicate which is more specific than the corresponding one in Q, query Q' can be an interesting substitute to Q since the other predicates must also be taken into account. For instance, if query Q = "A in {rabbit, hen} and B in {wheat, cabbage}" returns an empty answer, it is still possible that query Q' = "A in {rabbit} and B in {wheat, oats}" returns a non-empty one whereas the predicate on A is more specific in Q' than it is in Q. For a query Q' to be a possible substitute, it is necessary that Q' involves *at least one* predicate which is not as specific as the corresponding one in Q (but notice that if it were not the case, the answer to Q' would be empty – since the answer to Q is – and Q' would therefore not be in D^+).

Remark 3. The predicates from Q' which are strictly more specific than those from Q can be replaced by the latter ones.

The substitution process that we propose consists of the following three steps:

i) select the candidate queries (and adapt these queries, see algorithm below),

ii) compute the proximity degrees between the queries retained and the user query Q through the measure sbs3,

iii) determine the closest substitute to Q and process it.

Remark 4. For every predicate P from Q which is not "covered" by Q', i.e., which concerns an attribute on which there is no constraint in Q', one may compute the proximity between P and the entire domain of the attribute considered.

The conjunctive combination of the proximities related to the atomic predicates can be performed by means of a triangular norm, so as to obtain the overall proximity between two queries. Notice that alternative solutions could also be possible, for instance one might use the arithmetic mean. The substitution algorithm is outlined hereafter.

Algorithm:

Let Q' be a query from D^+ which shares at least one attribute from its "where" clause with that from Q. The five steps of the algorithm are:

i) replace the "select" clause from Q' by that from Q;

ii) remove from Q' every predicate that concerns an attribute absent from the "where" clause from Q;

iii) replace every predicate from Q' which is strictly more specific than the corresponding one from Q by the latter;

iv) for the other predicates, compute the proximity between the predicate from Q' and the corresponding one from Q, by means of measure sbs3, and replace the predicate from Q' by its union with that from Q. As to the predicates from Q which are not covered by Q', one computes their substitutivity degree relatively to the entire domain of the attribute involved, which boils down to replacing the predicate by "true";

v) aggregate the local proximities by means of a triangular norm (the idea is to assess the extent to which *every* predicate of the substitute query is close to the corresponding predicate from the initial failing query).

Example 4. Let Q be the following failing user query:

select #id **from** F **where** veg **in** {corn, rapeseed} **and** city **in** {Lannion, Caouennec, Prat} **and** area **in** [60, 100].

Let us assume that the domain of "veg" is:

{corn, rapeseed, sunflower, wheat, cabbage,
broccoli, potato, rutabaga}

and that the associated resemblance relation is:

	co	ra	su	wh	ca	br	po	ru
co	1	0.4	0.3	0.8	0.1	0.1	0.6	0.4
ra	0.4	1	0.9	0.6	0.2	0.2	0.1	0.1
su	0.3	0.9	1	0.5	0.1	0.1	0.3	0.3
wh	0.8	0.6	0.5	1	0.2	0.1	0.5	0.4
ca	0.1	0.2	0.1	0.2	1	0.9	0.6	0.7
br	0.1	0.2	0.1	0.1	0.9	1	0.4	0.6
po	0.6	0.1	0.3	0.5	0.6	0.4	1	0.8
ru	0.4	0.1	0.3	0.4	0.7	0.6	0.8	1

Let Q'_1 be the following query from D^+:

select #name **from** F
where veg **in** {wheat, rapeseed, sunflower} **and** city **in** {Lannion, Prat} **and** area = 125 **and** animal **in** {cow, pig}.

The query Q''_1 obtained by adapting Q'_1 according to the algorithm above is:

select #id **from** F
where veg **in** {corn, rapeseed, wheat, sunflower} **and** city **in** {Lannion, Caouennec, Prat} **and** (area **in** [60, 100] **or** area = 125).

The degree computed by sbs3 for the substitution of {corn, rapeseed} by {wheat, rapeseed, sunflower} equals:

(max(0.8, 0.6) + max(0.3, 0.9))/2 = 0.85.

Let us assume that the proximity over the areas is based on a fuzzy tolerance indicator Z with a triangular membership function of support [–50, 50]. The substitution of [60, 100] by 125 is assigned the degree 0.5 (i.e., the proximity degree between 100 and 125).

Finally, the degree computed for Q''_1 using the t-norm minimum is:

min(0.85, 0.5) = 0.5.

Let us now consider another query, denoted by Q'_2, from D^+:

select #name **from** F
where city = Caouennec **and** area **in** [80, 180] **and** animal **in** {sheep, goat}.

Altering Q''_2 according to the algorithm yields:

select #id **from** F
where city **in** {Lannion, Caouennec, Prat} **and** area **in** [60, 180].

As to the condition on "veg" we get:

sbs3({corn, rapeseed}, domain(veg)) =
(0.9 + 0.8 + 0.2 + 0.2 + 0.6 + 0.4)/6 = 0.52.

As to the condition on "area", we get:

sbs3([60, 100], [80, 180]) =
((150 – 100)/2)/(100 – 80) = 25/80 = 0.31.

Finally, the degree associated with Q''_2 is:

min(0.52, 0.31) = 0.31

and Q''_1 is a better substitute to Q than Q''_2.♦

Remark 5. In case of ties, one could take into account the cardinality of the result of each candidate query so as to break these ties, provided that these cardinalities are stored in D^+.

3 Fuzzy queries

Let us now move to the case where value constraints are expressed by means of fuzzy predicates. Let us consider a conjunctive fuzzy query $Q = P_1$ and ... and P_n where any predicate P_i is of the form "A_i is T_i" and T_i is a fuzzy term. Here, the fact that Q returns an empty answer means that there does not exist any element x in the database such that $\top_{i=1..n}(\mu_{Ti}(x)) > 0$, where $\top$ denotes a triangular norm generalizing the conjunction. This state of fact can be expressed by saying that the support of the query relatively to the database is empty.
In order to deal with this kind of queries, one needs to generalize measure sbs3 by replacing the arithmetic mean by a weighted mean, and by taking into account the resemblance between the degrees coming from the two fuzzy terms considered. The generalized measure obtained, which can also be seen as a variant of the interchangeability measure proposed in [5], is defined as:

sbs3(E, E') =
$[\Sigma_{x' \in \text{support}(E' - E)} w(x') * \sup_{x \in E} \min(\text{res}(x, x'), \Psi(\mu_{E'}(x'), \mu_E(x)))] / \Sigma_{x' \in \text{support}(E' - E)} w(x')$

where Ψ assesses the resemblance between two degrees in the unit interval — it can be defined e.g. as $\Psi(a, b) = 1 - |a - b|$ — and

$w(x') = \mu_{(E' - E)}(x') = \min(\mu_{E'}(x'), 1 - \mu_E(x'))$.

The weight w(x') captures the fact that it is all the more important to find a good substitute to x' as x' strongly belongs to E' – E. It is straightforward to show that if the sets are crisp, this formula reduces to that given in Section 2.
The definition above can be directly extended to the case of continuous fuzzy sets by replacing the sum by an integral.

Example 5. Let us consider the fuzzy sets:

E = {1/rapeseed, 0.8/cabbage, 0.3/wheat} and

E' = {0.4/rapeseed, 0.3/cabbage, 0.4/corn, 0.7/broccoli}.

With the most commonly used definition of the difference between fuzzy sets, i.e., $\mu_{(A - B)}(x) = \min(\mu_A(x), 1 - \mu_B(x))$, one gets:

E' – E = {0.2/cabbage, 0.4/corn, 0.7/broccoli}.

For "cabbage", the supremum equals:

sup(min(0.2, 0.3), min(1, 0.5), min(0.2, 1) = 0.5,

for "corn", it equals:

sup(min(0.4, 0.4), min(0.1, 0.6), min(0.8, 0.9)) = 0.8,

and for "broccoli" we get the degree 0.9.

Hence, the final substitutivity degree equals:

(0.2*0.5 + 0.4*0.8 + 0.7*0.9)/(0.2 + 0.4 + 0.7) = 0.81.♦

The algorithm given in Subsection 2.2 can be adapted straightforwardly. The notion of a "more specific fuzzy predicate" can be based on the inclusion between fuzzy sets proposed by Zadeh, i.e., $E \subseteq F \Leftrightarrow \forall x, \mu_E(x) \leq \mu_F(x)$.

4 Explaining the emptiness of the answer

Besides providing the user with a non-empty answer, we consider it important to also give him/her some explanations about: i) the reasons why the answer to his/her original query was empty, ii) the way his/her query has been modified so as to obtain a non-empty answer. The current section deals with the former aspect and gives the general principle of a mechanism for identifying (at least partly) the causes for the emptiness of the original answer, somewhat in the spirit of [20].

4.1 Boolean queries

Let us assume that one has available not only D^+ but also a repository D^- containing the failing queries submitted previously to the system. We suggest using both D^+ and D^- to identify some of the failing subqueries which cause the empty answer to the original user query.

Example 6. Let us consider the following failing query:

Q: veg in {corn, wheat, cabbage} and animal in {cow, pig},

and Q_1 and Q_2 two queries from D^+:

Q_1: veg in {corn, cabbage, rapeseed} and animal in {sheep, pig, goat},

Q_2: animal = pig,

as well as Q_3 a query from D^-:

Q_3: veg in {rapeseed, broccoli}.

From $Q_3 \in D^-$, one can deduce that there is no rapeseed in the database relation considered. From this result and $Q_1 \in D^+$, one infers that there are some farms growing corn or cabbage in the relation. From $Q_2 \in D^+$, one deduces that there are farms breeding pigs. From these two results, one can conclude that none of the terms from query Q leads to an empty answer. Consequently, the "minimal failing subquery" is Q itself, which means that there is some sort of "incompatibility" between (corn, wheat cabbage) and (cow, pig).♦

In order to provide the user with explanations, we suggest to partition the subqueries of Q into three classes: L_1 (those which are known to produce non-empty answers), L_2 (those which are known to be failing), L_3 (the others).
Let us first consider the case of set-based predicates. Let a query from D^- expressed as P_1 and ... and P_n. Every P_i is a condition of the type A_i in E_i, which corresponds to a disjunction ($A_i = v_{11}$ or ... or $A_i = v_{1p}$). The first step is to transform every query from D^- into a rule:

($A_i = v_{11}$ or ... or $A_i = v_{1p}$) and ...
and ($A_n = v_{n1}$ or ... or $A_n = v_{nk}$) → false.

Symmetrically, every query from D^+ gives birth to a set of rules:

($A_i = v_{11}$ or ... or $A_i = v_{1p}$) → true
...
($A_n = v_{n1}$ or ... or $A_n = V_{nk}$) → true.

These two types of rules constitute the rule base of the reasoning system used further.
Let us now consider a failing query Q. For each subquery of Q, one checks whether it is possible to infer true or false using the rule base. If it is possible to infer true, the subquery belongs to L_1, if one can infer false it belongs to L_2, otherwise it belongs to L_3. The subqueries have to be examined in increasing order of their size, starting with the atomic predicates, considering that a subquery including a failing subquery is itself failing (it is then useless to examine it). The corresponding algorithm, which exploits the concept of Minimal Failing Subqueries (MFS), is thus somewhat analogous to that proposed in [15], but here, we do not have to process any subquery to know if it is failing or not, due to the existence of the repositories D^+ and D^-.
In the case of intervals, one cannot replace a predicate by an explicit disjunction. One thus needs an inference engine able to deal with interval constraints.

4.2 Fuzzy queries

For discrete fuzzy sets, one uses the same principle as for Boolean queries. In the rules, the terms of a disjunction correspond to the values from the support of a fuzzy predicate.
For continuous fuzzy sets, one cannot express rules involving explicit disjunctions. A query $Q = P_1$ and ... and P_n from D^- produces a rule:

A_1 in support(P_1) and ... and A_n in support(P_n) → false.

and a query $Q = P_1$ and ... and P_n from D^+ gives birth to the set of rules:

A_1 in support(P_1) → true
...
A_n in support(P_n) → true.

Here again, the reasoning engine must be able to deal with interval constraints.

5 Related work

Some related work can be found in both domains of databases and information retrieval, including web search.

The *semantic caching* approach proposes to keep in a cache some previously executed queries along with their results and checks whether the answers to a given user query can be retrieved from the cache for optimization purposes. It uses the notion of *query containment* (i.e. a query Q is contained in a query G, if all answers to Q are also answers to G) to find the answers in the cache (see, e.g., [1, 7, 16, 19]). In a similar spirit, Ghosh *et al.* [14] propose a *query clustering* approach aimed at optimizing queries by reusing execution plans computed for similar queries.
On the other hand, the approach of *query rewriting* using views aims at finding view-based queries which are *equivalent to* (or *contained* in) a given user query. View-based query rewriting was first introduced using materialized views for query optimization purposes [6]. Afterwards, it was brought to the domain of data integration systems [2, 18] where the objective is to find *certain answers* to a query in a decentralized database context.
However, none of these approaches deals with the empty answer problem: they are concerned either in optimizing the access to information or in computing the set of certain answers to a query from distributed data sources. In our case, given a failing query, we are not interested in finding neither contained nor equivalent queries, since those would produce empty answers too.
In the information retrieval domain, other techniques are based on similarity measures and make use of previous executed queries to improve web search (see, e.g., [22, 24, 25]). The measures underlying these approaches are strongly based on relationships between keywords, whereas we deal with a more general type of conditions than those expressed by a set of keywords, namely value constraints.
There are also relevant work in the domain of *case-based reasoning,* which integrates past user experience to solve current queries. In particular, Fouqué *et al.*[10] propose to include additional information in the answer to a user query, using an approach that they call *associative query answering*. The basic idea is to extend the "select" clause of a user query with attributes which appeared in similar queries previously submitted by other users. As we do, the authors use a nearness measure between every pair of values of each attribute, which is used to evaluate similarity between the user query and those previously executed. However, that work does not deal with failing queries.
In contrast, Bidault *et al.* [3] tackle the *empty answer* issue and use a repository of predefined queries in the context of *mediation systems*. Although their approach shows some similarity with ours, their solution is still quite different. They build a set of predefined successful queries for each source to offer a substitute for a failing user query. Similarity degrees between the initial user query and predefined successful queries are computed on the basis of a hierarchy of concepts. Finally, some heuristics make it possible to select the "best" substitute to the initial user query. In this approach, the similarity degree concerns the extent to which two concepts share the same characteristics whereas the substitutivity measure we defined is based on the proximity between the domain values of the attributes involved in the queries.
Let us also emphasize that none of the aforementioned approaches deals with failing *fuzzy* queries.

6 Conclusion

In this paper, we have outlined an approach aimed at obviating empty answers to Boolean or fuzzy queries involving value constraints. The method proposed uses a query repository and is based on the adaptation of the past non-failing query which is the most similar to the user query considered. Moreover, a technique aimed at providing the user with some explanations about the emptiness of the result of his/her original query has been briefly outlined.

The perspectives for future work are manifold. First, it would be useful to tackle the implementation of the query repository and to devise access methods for retrieving the candidate queries as efficiently as possible. The logical formalism proposed in [16] to represent information in cache and the index implementation presented in [10] to access it could be of interest for that purpose. Secondly, it would be worth investigating the possibility of reusing the execution plans of past queries to optimize the evaluation of the selected substitute query, in a spirit similar to [14]. Another point worthy to study concerns the substitutivity measure which is at the heart of the approach. The measure advocated here should be compared with some others adapted from classical similarity measures, cf. [9]. Finally, we intend to generalize this approach to a broader class of queries, in particular those involving fuzzy joins.

References

[1] Y. Arens, C.A. Knoblock, Intelligent caching: selecting, representing and reusing data in an information server, Proc. of CIKM 1994, pp. 433-438, 1994.

[2] C. Beeri, A. Halevy, M.-C. Rousset, Rewriting Queries Using Views in Description Logics, Proc. of PODS'97, pp. 99-108, 1997.

[3] A. Bidault, C. Froidevaux, B. Safar, Similarity Between Queries in a Mediator, Proc. of ECAI 2002, pp. 235-239, 2002.

[4] P. Bosc, A. Hadjali, O. Pivert, Weakening of fuzzy relational queries: an absolute proximity relation-based approach, *Mathware & Soft Computing* vol. 14 (1) (2007) 35-55.

[5] P. Bosc, O. Pivert, On the comparison of imprecise values in fuzzy databases, *Proc. of the 6th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'97)*, pp. 707-712, 1997.

[6] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim, Optimizing queries with materialized views, Proc. of ICDE 1995, pp. 190-200, 1995.

[7] B. Chidlovskii, U.M. Borghoff, Semantic caching of web queries, VLDB Journal, vol. 9, pp. 2-17, 2000.

[8] W. Chu, H. Yang, K. Chiang, M. Minock, G. Chow, C. Larson, Cobase: A scalable and extensible cooperative information system, *Journal of Intelligent Information Systems*, 6(2-3), (1996) 223-259.

[9] V.V. Cross, T.A. Sudkamp, Similarity and Compatibility in Fuzzy Set Theory – Assessment and Applications, Physica-Verlag, Heidelberg 2002.

[10] G. Fouqué, W.W. Chu, H. Yau, A case-based reasoning approach for associative query answering, Proc. of ISMIS'94, pp. 183-192, 1994.

[11] T. Gaasterland, Cooperative answering through controlled query relaxation. *IEEE Expert*, 12, pp. 48-59, 1997.

[12] T. Gaasterland, P. Godfrey, J. Minker, An overview of cooperative answering, *Journal of Intelligent Information Systems*, 1(2), pp. 123-157, 1992.

[13] T. Gaasterland, P. Godfrey and J. Minker, Relaxation as a Platform for Cooperative Answering, Journal of Intelligent Information Systems, vol. 1, pp. 293-321, 1992.

[14] A. Ghosh, J. Parikh, V.S. Sengar, J.R. Haritsa, Plan Selection Based on Query Clustering, Proc. of VLDB 2002, pp. 179-190, 2002.

[15] P. Godfrey, Minimization in cooperative response to failing database queries. *Int. Journal of Cooperative Information Systems*, 6(2), 95-149, 1997.

[16] P. Godfrey, J. Gryz, Answering Queries by Semantic Caches, Proc. of DEXA 1999, pp. 485-498, 1999.

[17] S.Y. Huh, K.H. Moon, H. Lee, A data abstraction approach for query relaxation. *Information and Software technology*, 42, pp. 407-418, 2000.

[18] H. Jaudoin, J.-M. Petit, C. Rey, M. Schneider, F. Toumani, Query Rewriting Using Views in Presence of Value Constraints, Proc. of the Int. Workshop on Description Logics (DL'05), pp. 250-262, 2005.

[19] D.P. Miranker, M.C. Taylor, A. Padmanaban, A Tractable Query Cache by Approximation. Proc. of SARA 2002, pp. 140-151, 2002.

[20] A. Motro, SEAVE: A mechanism for verifying user presuppositions in query systems. *ACM Trans. on Off. Inf. Syst.*, 4(4), pp. 312-330, 1986.

[21] I. Muslea, Machine learning for online query relaxation. Proc. *Int. Conf. of Knowledge and Discovery and Data mining (KDD'04)*, pp. 246-255, 2004.

[22] V. Raghavan, H. Sever, On the Reuse of Past Optimal Queries, Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, United Stated, pp. 344-350 (1995).

[23] Z.W. Ras, D. Dardzinska, Failing queries in distributed autonomous information systems. *Proc. Int. symp. on Methodologies for Intelligent Systems (ISMIS'05)*, LNAI 3488, Springer-Verlag, pp. 152-160, 2005.

[24] J.R. Wen, J.-Y. Nie, H.-J. Zhang, Query Clustering Using User Logs, *ACM Transactions on Information Systems (ACM TOIS)*, Vol. 20 (1), pp 59-81 (2002).

[25] O.R. Zaïane, A. Strilets, Finding Similar Queries to Satisfy Searches Based on Query Traces. Proc. of OOIS Workshops 2002, pp. 207-216 (2002).